FPGA Prototyping of Asynchronous Networks-on-Chip

Jon Neerup Lassen

M.Sc. thesis
Thesis no.: 26
IMM, DTU
Kongens Lyngby 2008

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

Abstract

Network-on-chip (NoC) is an emerging paradigm for handling the communication in large system-on-chips. This project investigates the ability to prototype asynchronous NoCs on FPGAs.

The implementation of asynchronous circuits on standard FPGAs is highly experimental, therefore the first part of the project has been to establish a design flow for the implementation of asynchronous circuits on FPGAs. In the project an asynchronous best-effort NoC for an FPGA has been successfully developed. The NoC implementation consists of a router and network adapters and is implemented using a 4-phase bundled data handshake protocol. Cores connects to the network using an OCP interface. To demonstrate the NoC it has been implemented in a small multi-processor prototype using a mesh topology for the network.

Preface

This thesis has been carried out at the Computer Science and Engineering division of the Informatics and Mathematical Modelling department at the Technical University of Denmark from September 2007 to March 2008.

I would like to thank my supervisor Jens Sparsø for his guidance and support during the project. I would also like to thank Morten Sleth Rasmussen for his help.

Lyngby, March 2008

Jon Neerup Lassen

Contents

Abstract **i**

Preface **iii**

1 Introduction **1**

1.1 Project Description 1

1.2 Thesis Overview 2

2 Asynchronous Circuits on FPGAs **3**

2.1 Introduction 3

2.2 Asynchronous Circuit Design 4

2.3 Previous Work 8

2.4 FPGA Basics 9

2.5 Asynchronous Design Elements for FPGAs 12

2.6 Controlling Timing 23

2.7 Design Flow 33

3 Networks-on-Chip 41

3.1 Introduction to Networks-on-Chip 41

3.2 Basic Concepts 42

3.3 Previous Work 47

4 Asynchronous Network-on-Chip Design 49

4.1 General Network Design 49

4.2 Router Design 56

4.3 Network Adapter Design 64

4.4 Traffic Generator Design 73

5 Asynchronous Network-on-Chip Implementation 75

5.1 Router 75

5.2 Network Adaptor 78

5.3 Traffic Generator 79

6 Asynchronous Network-on-Chip Test 81

6.1 Introduction 81

6.2 FIFO 81

6.3 Input Port 82

6.4 Output Port 82

6.5 Router 83

6.6 Network Adaptor 84

7 Asynchronous NoC-Based MPSoC Prototype 85

7.1 Introduction 85

7.2 Synchronous NoC-Based SoC 85

7.3 MPSoC Overview 87

7.4 MPSoC Design 87

7.5 MPSoC Implementation 89

7.6 MPSoC Test 93

8 Discussion 95

8.1 Evaluation 95

8.2 Future Work 98

9 Conclusion 99

A Appendices 105

A.1 Perl SDF script 105

A.2 NoC Tests 107

A.3 MPSoC Tests 120

A.4 RPM Forum Post 124

A.5 VHDL Code 125

A.6 C-Code 261

Chapter 1

Introduction

1.1 Project Description

The scaling of microchip technologies has made it possible to fabricate large System-on-chip (SoC) designs. Network-on-chip (NoC) is an emerging paradigm for handling the global communication between subsystems in large SoC designs. Due to the scaling of microchip technologies the distribution of a global clock has become increasingly difficult. Designing the NoC using asynchronous design techniques is an appealing approach because it eliminates the need for a global clock. Several examples of asynchronous NoC implementations have been published. All of them are based on CMOS standard cells designs, which makes it complicated and expensive to build prototypes of NoC systems.

The purpose of this project is to investigate how to implement FPGA prototypes of asynchronous NoC systems. This will give researchers the possibility to perform experiments on different asynchronous NoC designs on an FPGA prototype and thereby avoiding to use a custom designed chip which is both expensive and time consuming to build. Because it is targeted at prototyping, reliability of the NoC is not a key concern. The primary goal is to develop a working system so emphasis has not been put on high performance or low cost.

The implementation of asynchronous designs on standard FPGAs targeted syn-

chronous design is highly experimental. The implementation presented in this thesis is mainly based on the experience collected in a few small projects carried out on IMM, DTU. The asynchronous FPGA design from these projects have been extremely simple; only small circuits that calculates the greatest common divider or generates a list of fibonacci numbers have been implemented. Thus a major part of this thesis is to establish a design flow for implementing large asynchronous systems on FPGAs.

1.1.1 Objectives

The objectives of the thesis are:

1. Establish a design flow for implementing asynchronous systems on FPGAs.
2. Develop a simple asynchronous best-effort NoC and implement in on an FPGA.
3. Develop an FPGA implementation of a multi-processor prototype with the asynchronous NoC used as interconnect.

1.2 Thesis Overview

The structure of the rest of this thesis is as follows:

Chapter 2 is dedicated to present the experiences learned about the implementation of asynchronous circuits on FPGAs. It is meant to present a general design flow for designing asynchronous circuits on FPGAs that is not specifically targeted at NoC design. It also includes an introduction to asynchronous design techniques.

Chapter 3 gives an introduction to NoC design and presents the previous work that have been used for the NoC design.

Chapter 4, 5, and 6 presents the design, implementation, and test of the developed NoC.

Chapter 7 presents a small prototype utilizing the developed NoC.

Finally chapter 8 and 9 contains the discussion and conclusion respectively.

Chapter 2

Asynchronous Circuits on FPGAs

2.1 Introduction

Asynchronous circuit design for FPGAs is not a straight-forward task. FPGAs are solely intended for synchronous designs, thus the design primitives available on the FPGA and the available design tools are only intended for synchronous designs. This chapter will give an explanation of what the challenges in asynchronous FPGA design are and how these challenges are overcome. The chapter is ended with a design flow guideline for implementing asynchronous circuits on FPGAs.

Section 2.2 will give a brief introduction to the fundamental concepts of asynchronous circuit design. Section 2.3 will present previous work about implementing asynchronous circuits on FPGAs. Section 2.4 will describe the FPGA that is used in the project. Section 2.5 will present the implementation of the basic asynchronous design elements. Section 2.6 will describe how timing is controlled when implementing asynchronous circuits. The last section 2.7 will give guidelines for the design flow for the implementation of asynchronous circuits on FPGAs.

2.2 Asynchronous Circuit Design

In traditional synchronous designs the flow of data is controlled by a global clock. In asynchronous design the flow of data is controlled locally between neighboring components using a request/acknowledge handshake protocol. The absence of a global clock gives asynchronous circuits some different properties compared to synchronous circuits. Some of the advantages are:

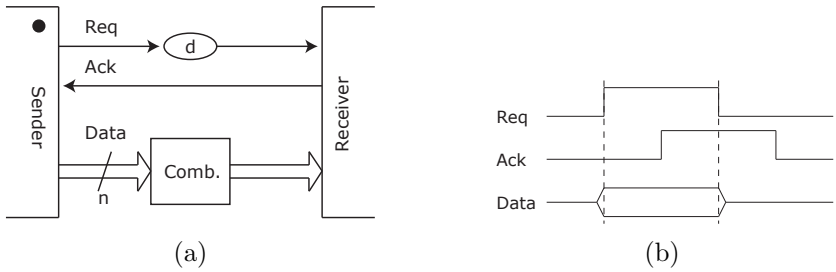
- *Low power consumption* – components are only active when they are actually used.
- *high operating speed* – the operating speed is not limited to the slowest component. The circuits will operate at their natural speeds.
- *Low EMC noise* – the local "clocks" tend to tick at random points in time.
- *No clock distribution/skew problems* – there is no clock!

The following sections will give a brief introduction to the fundamental concepts of asynchronous circuit design. For an in-depth presentation of asynchronous circuit design the reader is referred to [24], which also have been used as the source for the theory presented in the following sections.

2.2.1 Handshake Protocols

The handshaking between neighboring registers is carried out using a handshake protocol. The basic operation of a handshake protocol is: the sender sends a request to the receiver to inform that is has new data for it; when the receiver has captured the data, it acknowledges the request; and the sender is able to take its request down to be ready for another handshake. Two main types of handshaking protocols exists: *bundled-data* and *dual-rail*. In bundled-data protocols request and acknowledge uses separate signals, that are bundled with the data signal to form the handshake channel. In a dual-rail protocol the request signal is encoded into the data signals. In this project only the bundled-data protocol is used, thus dual-rail will not be presented here.

Figure 2.1(b) shows an example of the 4-phase bundled-data protocol. The sender sets the data signals and asserts the request signal. The receiver reads the data and responds by asserting the acknowledge signal. When the receiver sees that the acknowledge signal has been asserted, it pulls down the request signal. The receiver ends the transaction by pulling the acknowledge signal down.



(a) (b)

Figure 2.1: The 4-phase bundled data protocol.

Note that the request and acknowledge signal must return to zero before the transaction ends. A more efficient *2-phase bundled-data* protocol exists where the superfluous return-to-zero transition is avoided. In the 2-phase protocol a request or acknowledge event is encoded as a signal transition on the control wire, e.g. a $0 \rightarrow 1$ or a $1 \rightarrow 0$ transition, in contrary to the 4-phase bundled-data where a request or acknowledge event is encoded by the level of the respective control wire.

Depending on if it is the receiver or it is the sender who initiates the transaction, handshake channels can be grouped into another two types: *push channels* and *pull channels*. In push channels the sender initiates the transaction by sending a request to the receiver. The request signal tells the receiver that the sender has data for it. In pull channels the roles are interchanged, i.e. the receiver initiates the transaction using the request signal, and the request tells the sender that it is ready to receive data. To distinguish between pull and push channels the initiating part is marked with a dot on the diagram as shown on figure 2.1(a).

All bundled data protocols have the timing requirement that the sequence of events at the sender's side is preserved at the receiver's side. For a 4-phase bundled-data push channel this means that the designer *must* assure that the the receiver sees valid data before the request is asserted. If the data signals are delayed, e.g. by propagating through combinatorial logic, the request signal must also be delayed accordingly. This is referred to as *delay matching*. To delay a signal a delay element is used. In figure 2.1(a) a delay element is inserted on the request signal. The inserted delay must at least match the delay through the combinatorial circuit that the data signals propagates through.

The time interval in which data is valid during the handshaking phase is described by the *data validity scheme*. For at 4-phase bundled-data channel four different data validity schemes exists: early, broad, late, and extended early.

- Early data validity: data are valid from the rising request event to the rising acknowledge event.
- Broad data validity: data are valid from the rising request event to the falling acknowledge event.
- Late data validity: data are valid from the falling request event to the falling acknowledge event.
- Extended early data validity: data are valid from the rising request event to the falling request event.

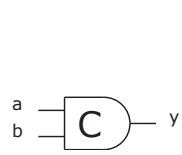
The choice of data validity scheme affects the implementation of the handshaking components.

In synchronous designs signals are only required to carry the correct value during a well defined period around clock-ticks. In between clock-ticks the signals may exhibit hazards or transitions. In asynchronous designs this is not allowed because all signal transitions have a meaning. For example, a hazard on an acknowledge signal will make the sending circuitry believe that the receiver already has captured the data, even though this is not the case. Consequently asynchronous circuits requires that all control signals must be valid and hazard free at all times.

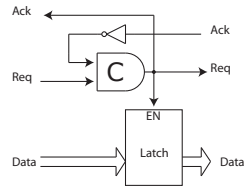
2.2.2 The Muller C-Element

To be able to design hazard free control citcuits a new component is needed: the Muller C-element. The C-element has the property that it indicates both when all inputs are low and when all inputs are high. In comparison a conventional AND gate only indicates when all inputs are high and a conventional OR gate only indicates when all inputs are low.

The Muller C-element is a state holding component which is 0 if both inputs are 0 and 1 if both inputs are 1. If the inputs are 01 or 10 the C-element will keep its previous state. Figure 2.2 shows the gate symbol and the truth table for the C-element. The use of the C-element in a handshake component is shown in figure 2.2 (c). This circuit is a single stage of the Muller pipeline, which is the backbone of almost all asynchronous control circuits.

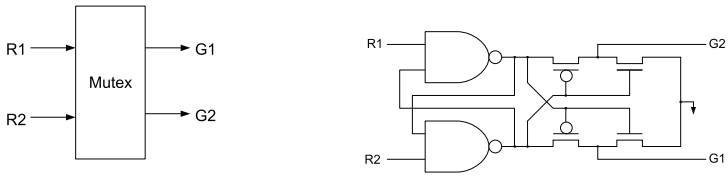


a	b	y
0	0	0
0	1	no change
1	0	no change
1	1	1



(a) (b) (c)

Figure 2.2: The Muller C-element: (a) gate symbol, (b) truth table, and (c) a Muller style handshake latch.



(a) (b)

Figure 2.3: Mutex component: (a) symbol, and (b) possible implementation (from [24]).

2.2.3 Mutual Exclusion

Handshake components with more than one input channel usually requires that the input requests are mutual exclusive, i.e. only one request is high at a time. Since the requests may arrive at exactly the same time a mutual exclusion (mutex) component is needed. Figure 2.3 shows the mutex symbol and a possible CMOS transistor level implementation (from [24]). The mutex should exhibit the following behavior: If only one request is asserted the corresponding output should be asserted. If both inputs are asserted but one of them is asserted before the other, the late request should be held back and only allowed to propagate when the other request has been taken down. If both request are asserted at the same time, the mutex must make an arbitrary decision of which signal should be allowed to propagate first. A possible implementation of a mutex component has two cross-coupled NAND-gates, which enables one input to block the other. If two requests arrives simultaneously the cross-coupled NAND-gates will become metastable, hence a metastability filter is needed at the outputs. The shown implementation of the metastability filter is a CMOS transistor level implementation. In section 2.5.2 a metastability filter that can be implemented in an FPGA is presented.

2.3 Previous Work

The previous work about implementing asynchronous circuits on FPGAs is very limited. A number of special courses and course projects (from the course 02204 – Design of Asynchronous Circuits) supervised by Prof. Jens Sparsø have investigated the implementation of basic asynchronous design elements. The *02204 course project* by Knud Hansen and Guillaume Saoutieff [11] is the first project. A LUT based C-element is implemented together with a fork, a join, a merge, a mux, and a demux component. A simple circuit computing the GCD (greatest common devisor) is implemented on a Xilinx Spartan-II FPGA. All components are based on the 4-phase bundled-data handshake protocol. In a later *02204 course project* by Tue Lyster and Morten Thomsen [15] an asynchronous symbol library for the Xilinx schematics editor (Xilinx ECS) based on the components created in [11] is created. In the special course project *Asynchronous Circuits in FPGA* by Mikkel Stensgaard [26] a number of improvements and additions have been made. The implementation of the components presented in [11] has been improved to better fit the anatomy of an FPGA. The delay element is now implemented as a chain of AND gates. A design flow for implementing Petrify circuits is presented. Un-, semi- and fully-decoupled latch controllers and mux and demux components are specified by STGs and implemented using Petrify. The latch controllers are tested in a FIFO and in a FIFO-ring circuit. Again the GCD circuit is used as test circuit for the other components. All components are added to a VHDL library. The circuits have been implemented on a Xilinx Spartan-IIE FPGA. In the special course *Asynchronous Circuits on FPGAs* by Morten Rasmussen, Christian Pedersen, and Matthias Stuart [21] the implementation of the components from [26] is changed to fit a new VHDL library. The library is extended with 4-phase dual-rail implementations of the components from [26]. The new library allows for easy switching between the two types of handshake protocols. The following new components are added: adder, subtracter, inverter, shifter, and comparator. Also, the library is documented in a complete library reference. The library utilizes user-defined data types which must be converted by wrappers for successful implementation. In the special course *Implementation of Asynchronous Circuits in FPGAs* by Esben Hansen and Anders Tranberg-Hansen [10] another complete redesign of the library has been carried out after evaluation of the existing library from [21]. They found that the use of user-defined data-types made it too tedious to implement even simple circuits. New 4-phase bundled data components are added: a register file, a block-ram based memory, a AND-, OR-, NOR-, and a XOR- component, and a simple ALU. The components are tested in a simple Fibonacci circuit on a Spartan-3 FPGA. Also, oscilloscope measurements of the delay element is performed. A user guide for using the library is included along with a complete library reference. In the 02204 course project *FPGA Implementation of an Asynchronous Arbiter* by Mads Kristensen and Jon Lassen [14] a mutex and an

arbiter component is implemented. The mutex is implemented solely in LUTs and it is based on a standard gate mutex design presented by Ran Ginosar [8]. The design of the arbiter is based on the design from [24] and is implemented on a Xilinx Spartan-3 FPGA.

In the Aspida project [13] made by a consortium between FORTH-ICS, Politecnico di Torino, University of Manchester, and IHP Microelectronics a *de-synchronized* implementation of the DLX RISC CPU is presented. The DLX RISC CPU is a 5-stage pipelined CPU similar to the MIPS processor. De-synchronization is a method for converting an existing synchronous design into an asynchronous systems. When de-synchronization is performed all pipeline flip-flops are taken out and replaced by latches and asynchronous control circuits. The asynchronous pipeline latches are implemented so they are guaranteed to provide an equivalent behavior as the clocked flip-flops. This is done without touching the datapath at all. In this way the global clock is completely replaced by handshake signals. Delay elements must be inserted on the request path to match the delay of the combinatorial blocks between the asynchronous pipeline latches. The processor has been implemented on a Xilinx Spartan-2E FPGA and on a chip.

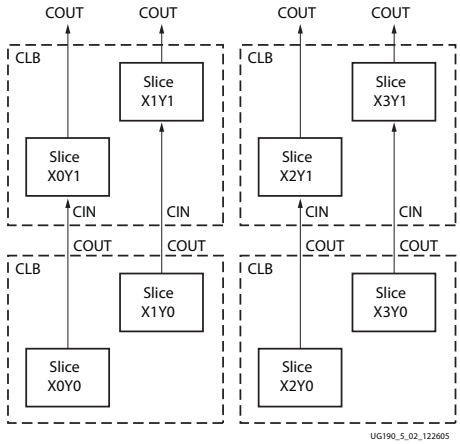
Details from the set of previous work presented here, which are interesting for this project, is presented in the relevant sections in the report.

2.4 FPGA Basics

This section will give a short introduction to the Xilinx FPGA used in the project and the development tools provided by Xilinx.

For the project the XC5VSX50T Xilinx Virtex-5 FPGA is used. The Virtex-5 is the newest FPGA generation supplied by Xilinx. The description of the FPGA is focused on how the logic resources are organized, because it is the most interesting from an asynchronous design point of view.

The FPGA consists of a large array of *Configurable Logic Blocks* (CLBs). Each CLB is connected to a *switch matrix* which handles the routing between the CLBs. A CLB contains two *slices* placed in separate columns. The slices does not have any direct connection between them, but each slice has a carry-chain which connects slices in the same column. Figure 2.4 shows the row and column relationship between CLBs and slices and the slice numbering scheme. The slice numbering is important for RPM creation, which is described in section 2.6.3.



(b)

Figure 2.4: Arrangement of CLBs and slices, from [35].

Each slice contains four Look-Up Tables (LUTs), four storage elements, multiplexers and carry-logic. The LUTs are used as logic functions generators and have 6 inputs and two outputs. The extra output allows the LUT to perform two different logic functions, if the functions have common inputs. The storage elements can be configured to behave either as a latch or as a flip-flop. In the asynchronous design components presented later in this chapter, the LUTs are also used as state-holding elements by feedback-coupling the output.

The FPGA has a total of 32640 LUTs and the same number of flip-flops/latches. Earlier generations of Xilinx FPGAs only had 4-input LUTs, thus with 6-input LUTs more logic can be packed into fewer LUTs.

The ISE software package is the logic design environment provided by Xilinx. Below is a description of the most important ISE tools which have been used during the project:

Project Navigator is the primary user interface for ISE. Most other tools can be accessed from here.

XST is the Xilinx synthesizer. Performs the logic synthesization of the VHDL to Xilinx specific netlist files.

MAP performs the mapping from the synthesized netlist to FPGA primitives.

PAR performs place and route of the mapped design.

Floorplanner used to perform floorplanning tasks. It can be used before MAP and after PAR. Before MAP it is used to assign constraints to the design. After PAR it can be used to manually make changes to the floorplan. It can also be used in an iterative process of re-assigning constraints and rerunning MAP and PAR.

FPGA Editor can be used to manually fine-tune the design after PAR. It can also be used as a detailed viewer of the place and routed design.

Design constraints are used to constrain the final implementation produced by the tools, e.g. tell the tools to place two logic functions in the same slice. Constraints can be added in two ways: Directly in HDL or in the User Constraints File (UCF). Constraints added in the UCF file is not read until after synthesis. Not all constraints can be added in HDL. The Xilinx Constraints Guide [29] documents all the available constraints.

Simulations of the design can be performed on four different levels of abstractions:

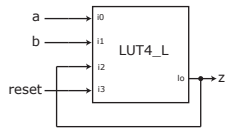
Behavioral simulation is an RTL level simulation of the design. It is used to validate correct functionality of the design. No timing information is included, so all signals changes instantaneously.

Post-Translate simulation is a gate-level functional simulation of the synthesized design. Is used to verify that the design has been synthesized correctly. Still no timing information is included.

Post-MAP simulation is run after MAP and provides partial timing information. The simulation includes gate delays but no routing delays. It is primary used as a debug step if Post-PAR simulation fails.

Post-PAR simulation provides full timing information. It simulates the design after place and route and contains both gate and routing delay.

For the Behavioral simulation FPGA primitives is simulated using a library called UNISIM while after synthesis the SIMPRIM library is used. The SIMPRIM library uses a more detailed model of the primitives. For asynchronous design the primary simulation modes used is the Behavioral and Post-PAR.



(a)

reset	z	b	a	z
0	x	x	x	reset_value
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(b)

Figure 2.5: C-element LUT implementation and truth table

2.5 Asynchronous Design Elements for FPGAs

Section 2.2 presented the fundamental concepts of asynchronous circuit, where a number of asynchronous design elements was presented. This section will present FPGA implementations of these basic building blocks along with a synchronizer component.

2.5.1 C-Element

The C-element is a simple state holding device much similar to a set-reset latch. The truth table was shown in figure 2.2 (p. 7). The implementation presented here is from the asynchronous circuit FPGA design library presented in [10] and it has not been changed for the use in this project.

The C-element can be implemented in a single LUT primitive with the output looped back to one of the inputs. This is shown in figure 2.5. A *generic* value is used to define the desired reset value for proper initialization. The instantiated LUT is a `lut4_l` primitive which is a LUT with local output. This instructs the tool to use local routing for the feedback signal.

In figure 2.6 an example of a VHDL instantiation of a C-element is shown. The truth table values from figure 2.5(b) is used as the initialization value. The implementation of the C-element is found in appendix A.5.1.2 (p. 127).

```
c_element: lut4_l
  generic map (
          init => "11101000" & reset_vector
  )
  port map (
          i0 => a,
          i1 => b,
          i2 => s_out,
          i3 => reset,
          lo => s_out
  );
```

Figure 2.6: VHDL instantiation of a C-element, from [10]

2.5.2 Mutex

The mutex component was introduced in section 2.2.3 and figure 2.3 (p. 7) showed a possible implementation of mutex. As shown on the figure a metastability filter is needed on the output to prevent the circuit from propagating possible undefined values, resulting from a metastable state at the cross-coupled NAND gates. An FPGA implementation of a mutex component is presented in [14] with satisfactorily results. This implementation has been used for this project. The VHDL code for the mutex implementation is found in appendix A.5.1.3 (p. 128).

The following will be presented in this section:

- The implementation of the mutex from [14].
- Some small modifications to the implementation to optimize it for a Virtex-5 FPGA.
- A solution to post place and route simulation problems of the mutex that has not been covered in [14].

An FPGA implementation of the mutex can (of course) only use the primitives available on the FPGA. The metastability filter in figure 2.3 is a CMOS transistor level implementation, thus it cannot be implemented in an FPGA. In [8] Ran Ginosar presents a mutex component build only from standard gates. The standard gate mutex design is shown in figure 2.7. The design still uses two cross-coupled NAND gates to let one input block the other. The metastability filter is implemented by two AND gates with one inverted input. Each of the four gates can be implemented in one LUT primitive.

The circuit cannot be considered as a safe design; if the NAND gates gets into metastability, they will stay there for an unknown length of time, but will

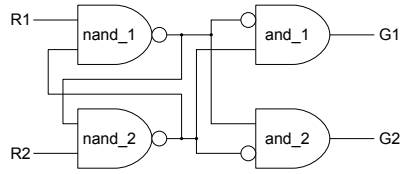


Figure 2.7: A mutex component build from standard gates

eventually choose one side randomly. While the NAND gates are in a metastable state, the AND gates will have unspecified behavior, because their inputs are undefined. However, If the NAND gates stabilizes "fast enough", the AND gates will not "see" the metastability for a long enough period to propagate the undefined inputs. To assure that the NAND gates stabilizes as fast as possible, they should be placed in the same slice to minimize the routing delay.

Another reason to place the NAND gates in the same slice, is to optimize the fairness of the mutex. The fairness is very dependant on the wire delays between the gates. If the wire delay of the cross-coupling signal from NAND_1 to NAND_2 is larger than the wire delay from NAND_2 to NAND_1 the $R2$ will get higher priority than $R1$, since the NAND_1 gate will be blocked faster. To make the implemented mutex as fair as possible the wire delays between the two nand-gates should be kept as equal as possible.

The mutex presented in [14] is implemented on an older FPGA generation with only two LUTs in each slice, so the mutex occupies two slices. Therefore the implementation has been changed slightly to fit the mutex in a single slice. Everything else is unchanged.

In the implementation of the mutex the four gates are placed in the same slice using `rloc` constraints (further explained in section 2.6.3). This will keep the wire delays between the gates as equal as possible. However it is not possible to specify the exact placement within the slice, hence some variations in the wire delays may occur. In an actual example from a post place and route simulation, the wire delay from NAND_1 to NAND_2 is 186 ps while the wire delay from NAND_2 to NAND_1 is only 130 ps. In this example the $R2$ signal will have priority, however the priority may be different when implemented on a FPGA since the relation between the delays may be different for an actual circuit. The small delay difference internal in the mutex component will most likely be insignificant compared to the difference in wire delay experienced by the input signals.

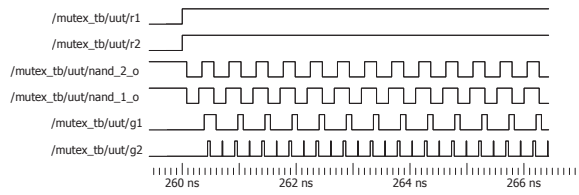


Figure 2.8: Printout from Modelsim showing an oscillating mutex.

The mutex has not been analyzed for Mean-Time-Between-Failure (MTBF). The theory for determining the MTBF of the mutex is the same as for the synchronizer which will be presented in section 2.5.4. In fact a synchronizer is a special case of a mutex, where the clock is connected to one of the inputs [20]. Since this project is aiming at system prototyping and not at in-production systems, the standard gate mutex is used without any further analysis or testing for MTBF and fairness.

There exists some issues with simulation of the mutex after place and route that has not been covered in [14]. In an actual circuit the NAND gates will not stay in a metastable state forever. This situation is different when it comes to simulation. During simulation the metastable state will result in an infinite oscillation between 0 and 1. In a behavioral (RTL) simulation the simulation will stop due to the oscillation. This happens because the simulation cannot proceed to the next delta-time and an *iteration limit reached* error is issued. During a post place and route simulation the oscillation will propagate to the outputs with a period matching the wire- and gate-delays. Figure 2.8 shows this situation. The period of oscillation is 476 ps for all oscillating signals which matches with the wire and gate delays of the simulation model.

In the case of a behavioral simulation the problem is easily solved by using a higher-level (non-synthesizable) simulation model of the mutex. This solution is used in [14].

In the case of a post place and route simulation the solution is not so easily solved. If the design hierarchy is kept all the way from synthesis to place and route it will also be possible to insert a strictly behavioral simulation model of the mutex into the post place and route simulation model. But if the design is flattened during synthesis it will be very tedious to insert another simulation model. Also, the timing behavior of the mutex will be lost. Therefore another

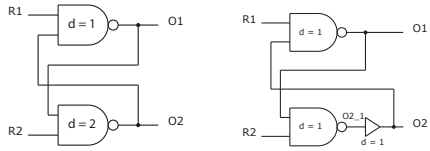


Figure 2.9: NAND stages of an unfair mutex. (a) shows the desired NAND stage. (b) shows the possible FPGA implementation of the circuit.

solution is needed. Two other solutions have been considered:

- Implementation of an unfair mutex.
- Make the implemented mutex unfair, by changing the simulation model.

Both solutions tries to break the oscillation by making the gate delay of one of the NAND gates larger than the other. By only changing the simulation model some inconsistency will be introduced between the actual circuit and the simulated circuit. If the changes made have minimal influence on the timing behavior of the mutex this inconsistency can be neglected.

The delay model used in the SIMPRIM simulation library effects how the mutex simulation problem can be solved. In VHDL delays can be modeled in two ways: as transport delays and as inertial delays. A transport delay models an ideal device with infinite frequency responses, where any input pulse will produce an output pulse. An inertial delay models devices with finite frequency responses, where an input pulse must have a minimum length before an output pulse is produced, otherwise it will be rejected. By studying the source code of the SIMPRIM simulation library it can be seen that the delay model for wire and gate delays are specified in a library called VITAL (VHDL Initiative Towards ASIC Libraries) which models the delays as transport delays. A simple solution could be to change the delay model used in the library to inertial delays. This will however affect the simulation of all components in the design, which is not desirable.

The first solution considered is the implementation of an unfair mutex. An unfair mutex should have unequal gate delays of the NAND gates. This will give the fast gate priority over the slow gate. In figure 2.9(a) this situation is illustrated with gate delays of 1 and 2 respectively. The LUT primitives in an FPGA all have the same timing characteristics, therefore it is only possible to imitate a slow gate as a concatenation of two gates, as shown in in figure 2.9(b).

```
(CELL (CELLTYPE "X_LUT6")
 (INSTANCE nand_1)
  (DELAY
   (ABSOLUTE
    (PORT ADR3 ( 914 )( 914 ))
    (PORT ADR4 ( 130 )( 130 ))
    (PORT ADR5 ( 1013 )( 1013 ))
    (IOPATH ADR3 O ( 80 )( 80 ))
    (IOPATH ADR4 O ( 80 )( 80 ))
    (IOPATH ADR5 O ( 80 )( 80 ))
   )
  )
 )
```

(a)

```
(CELL (CELLTYPE "X_LUT6")
 (INSTANCE nand_1)
  (DELAY
   (ABSOLUTE
    (PORT ADR3 ( 914 )( 914 ))
    (PORT ADR4 ( 0 )( 0 ))
    (PORT ADR5 ( 1013 )( 1013 ))
    (IOPATH ADR3 O ( 80 )( 80 ))
    (IOPATH ADR4 O ( 0 )( 0 ))
    (IOPATH ADR5 O ( 80 )( 80 ))
   )
  )
 )
```

(b)

Figure 2.10: Delay specification of a 2-input NAND gate with reset from the simulation SDF file. (a) original and (b) is modified to decrease the delay for the ADR4 port.

Due to the transport delay model used in the SIMPRIM simulation library the circuit in figure 2.9(b) will still oscillate, because all pulses on the O2_1 signal will propagate to the the O2 signal. Consequently it is not possible to solve the simulation problem by implementing a simple unfair mutex.

The chosen solution to solve the oscillation problem is to alter the post place and route simulation model. The post place and route simulation model consists of two files: an VHDL *netlist* file and an *SDF* file. The VHDL netlist instantiates simulation models of the FPGA primitives from the Xilinx SIMPRIM library. The SDF file specifies all wire and gate delays used in the simulation. The format of the SDF file is specified using the *Standard Delay Format Specification* [18]. In figure 2.10(a) an example of a delay specification for a NAND gate with a reset input is shown. Wire delays are modeled as delays at the input ports and is specified as `PORT` delays. Gate delays are specified as `IOPATH` delays. Both wire and gate delays can be specified individually for each input. A delay is specified as the rising and falling delay for the particular input and the unit is *ps*.

To solve the oscillation problem one of the NAND gates should be made faster than the other by decreasing the `PORT` and/or the `IOPATH` delays in the SDF file. It is only necessary to decrease the delay of the specific input connected to the other NAND gate; the other inputs can be leaved untouched. This will make the propagation delay through entire mutex element unaffected by the delay change. How much should the delay be decreased to kill the oscillation? Because the transport delay model is used in the simulation model the combined wire and gate delay through the gate must be 0 before the oscillation is killed. In figure 2.10(b) the modified SDF delay specification is shown. Figure 2.11 shows a simulation of the mutex after modification of the SDF file.

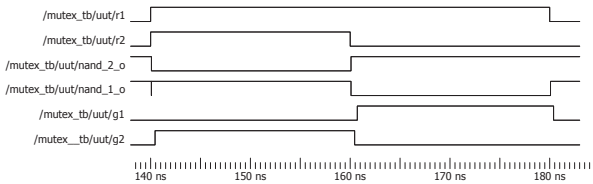


Figure 2.11: Simulation of the mutex after modification of the SDF file.

A Perl script that modifies all instances of NAND pairs in an SDF file as described above has been written and can be found in appendix A.1 (p. 105).

2.5.3 Delay Elements

In asynchronous circuit design the ability to delay a signal in a precise and predictable manner is crucial. When performing delay matching of an asynchronous circuit a delay element is inserted in the request path to delay the request signal by an equal amount of time compared to the delay experienced by the data signal, or to put in another way: the *minimum* delay of the delay element should at least match the *maximum* delay experienced by the data signals. When designing traditional synchronous circuits the maximum allowed clock frequency of a design is solely determined by the *maximum* delay through the combinatorial circuit, i.e. synchronous designs are inherently insensitive to the minimum delay of a combinatorial circuit.

In the datasheet for the Virtex-5 FPGA [34] the maximum delay through a LUT is specified to be between $0.08ns - 0.10ns$ [1], but the minimum delay is unspecified. The only guarantee about minimum delays given by Xilinx is that hold times are never violated. In general minimum delays in CMOS designs are usually not very well defined, since there can be large variations with e.g. change of temperature, supply voltage, etc. In an answer to a question posted in a newsgroup (dated 1996) [1] an Xilinx employee estimates that the minimum delay through a LUT approximately will be 25% of the specified maximum delay. It has not been possible to find any official estimates from Xilinx. This ratio between minimum and maximum delays are given for variations in supply voltage, temperature, and processing, so the delay difference between two LUTs on the same chip, under the same operating conditions, must be expected to be much lower. In this project no incidents have been encountered where a design have failed due to the aforementioned delay variations. The problems may be more prominent if the designs are tested on more different FPGAs and under varying operating conditions.

[1] Varies with the *speedgrade* of the FPGA

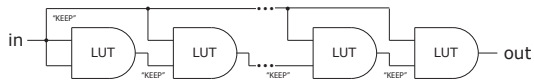


Figure 2.12: Asymmetric delay element.

A circuit using the 4-phase bundled data handshake protocol can be designed such that, it is only necessary to insert delays on the rising edge of the request signal. Delays on the falling edge will only slow down the circuit. An asymmetric delay element with this property is shown in figure 2.12. A transition from high to low will have to propagate through the entire chain of AND gates, while a low to high transition only have to propagate through the last AND gate. The signal will be delayed by the combined amount of gate and routing delay in the LUT chain.

In the rest of this section the following points will be presented:

- The implementation of the delay element presented in one of the special course projects [10].
- The implementation of the delay element used in Aspida [13]
- The implementation of the delay element used in this project.

In [10] an FPGA implementation of an asymmetric delay element is presented. The implementation instantiates a chain of LUT-instantiated AND gates connected as in figure 2.12. The number of AND-gates in the delay element is parameterized. To avoid that the synthesizer optimizes the LUT-chain away the `keep` constraint is applied to the signals connecting the gates. The `keep` constraint is a synthesis and mapping constraint that tells the synthesizer and mapper not to merge the two components connected by the signal into one component, thus keeping the signal in the design.

The design of the delay element used in Aspida project [13] is a little different than the one presented in [10]. It consists of two parts: a symmetric part and an asymmetric part. The symmetric part is used to generate a pulse delay and consists of a chain of a even number of inverters. The pulse delay is used to control the pulse width of the latch control signal. The asymmetric part is used to generate a matched delay and consists of a chain of AND gates similar to the one in figure 2.12. They also use the `keep` constraint to avoid that the synthesizer optimizes the delay element away. In the delay element used in the Aspida project [13] they experience a "`keep` conflict" error when the `keep` constraint is assigned to two signals which in fact are the same signal. This happens

with the first AND gate in the LUT-chain. They solved this issue by inserting two inverters in front of the first AND gate. To improve the predictability of the delay element, they manually restrict the physical placement of each delay element to a specific area of the FPGA by applying a constraint called `area_group` using the Floorplanner tool. By constraining the placement of the delay elements they experience improved predictability without using extensive floorplanning. They also observe increased predictability when the available area is small and decreased predictability when the available area is increased. The other option they have tried is to manually assign each LUT in the delay element to physical slice placement using the `loc` constraint. They claim that when the `loc` constraint is used, the predictability of the delay is nearly 100%. However, it turned out that the use of `loc` constraints had a very negative impact on the optimization of the datapath, especially when the utilization of the FPGA resources was high. Their conclusion is that the use of the `area_group` constraint gives almost the same predictability, as when `loc` constraints are used, and it requires less floorplanning and it does not have the optimization issues of the datapath experienced with the `loc` constraint.

The implementation of the asymmetric delay elements used in this project is a modified version of the asymmetric delay element presented in [10]. The implementation is modified by constraining the placement of the LUTs in the delay-chain to improve predictability. Constraining the placement will minimize variations in the routing delay, and thereby improve the predictability. The VHDL code for the delay element is found in appendix A.5.1.1 (p. 125).

A different approach is used for constraining the placement of the delay elements, than the one used in Aspida. Instead of constraining the delay LUTs to a physical area of the FPGA, only the relational placement between the LUTs in the delay element are constrained. This allows the tool to place the complete delay element anywhere on the FPGA area, while maintaining the internal placement of the LUTs in the delay element. This is done by assigning `rloc` constraints to the LUTs. A component constrained using `rloc` is referred to as an relationally placed macro (RPM) in the Xilinx documentation. The use of RPMs is explained in more detail in section 2.6.3

The layout of the delay LUTs is shown in figure 2.13. The delay LUTs are placed such that the signal between two consecutive LUTs in the LUT-chain will have to be routed to the neighboring CLB in the vertical direction. The main reason for creating the delay element as an RPM is to improve the predictability, however placing the delay LUTs such that longer routing path is required will improve the performance of the delay element, i.e. increasing the delay without using additional LUT resources. Only a limited experimentation of different placement layouts have been tried. If the layout in figure 2.13 is changed, such that the routing is done in the horizontal direction instead of in the vertical direction, the

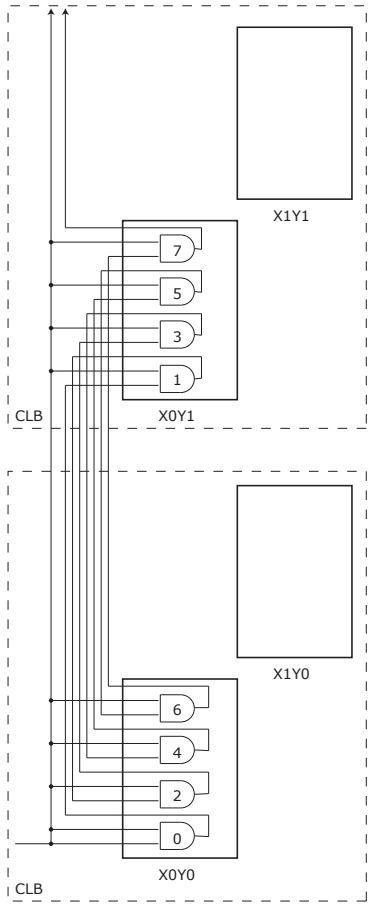


Figure 2.13: Arrangement of delay LUTs.

tool will issue an error, that the routing resources between the CLBs have been exhausted. Hence, a more optimal placement may exist, but if the utilization of routing resources is near saturation the performance of neighboring logic may be affected.

The issues with `keep` conflicts experienced in Aspida have not been experienced in this project. The version of the XST synthesizer that is used in this project automatically solves `keep` conflicts. However, it has been observed that the synthesizer will optimize the first AND gate into a simple buffer LUT. This optimization does not change the intended function of the LUT-chain since the signal still have to propagate through the LUT.

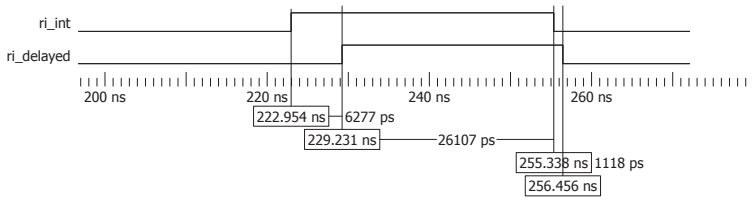


Figure 2.14: Modelsim print of a delay element simulation of size 10 showing the $0 \rightarrow 1$ and $1 \rightarrow 0$ delay.

Figure 2.14 shows a print of a Modelsim simulation of a delay element with a size of 10. The asymmetric properties are clearly shown with a low $\rightarrow$ high delay of 6.3 ns and a high $\rightarrow$ low delay of 1.1 ns. In section 2.6.1 a number of experiments of the size and predictability of the delay element in different contexts are presented.

2.5.4 Synchronizer

When a synchronous system communicates with the outside world it must use a synchronizer circuit. All inputs to the system that does not come from the same clock domain must be passed through a synchronizer to assure proper synchronization with the local clock-domain. The synchronizer will assure that the input signal satisfies the setup and hold time requirements of the local clock-domain. The problem with synchronization is well-known and described in many textbooks on digital design, e.g. in [27]. In a GALS (Globally Asynchronous Locally Synchronous) design with several local clocked synchronous circuits connected by an asynchronous interconnect, such as the system presented in chapter 7, a synchronizer is needed on the signals coming in from the interconnect. The most common synchronizer design is to let the asynchronous signal pass through a series of flip-flops clocked with the clock of the synchronous system. This is also the method applied in this project. Figure 2.15 shows a synchronizer design with two flip-flops.

A synchronizer will always suffer from metastability problems. If the asynchronous input changes during the decision window of the flip-flop the output of the flip-flop may become metastable and stay in the metastable state for an arbitrary period of time. By having more concatenated flip-flops in the synchronizer the probability that the output of the synchronizer becomes metastable can be reduced, however it can never be removed completely. In the Xilinx Application Note *Metastable Recovery in Virtex-II Pro FPGAs* [2] the MTBF of

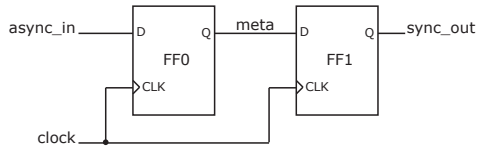


Figure 2.15: Synchronizer design with two concatenated flip-flops.

a synchronizer flip-flop is measured for a Virtex-II Pro FPGA. The conclusion is that if a two flip-flop synchronizer is used the metastable delay can safely be ignored for speeds below 200 MHz. It also states that for this conclusion to hold, the routing delay between the two flip-flops should be minimized. The MTBF is a statistically defined value and is calculated by the following formula:

$$MTBF = \frac{e^{K2 \cdot \tau}}{F1 \cdot F2 \cdot K1}$$

where $F1$ is the frequency of the clock input of the flip-flops, $F2$ is the frequency with which the asynchronous input changes, $K1$ is a device dependent constant describing the likelihood of going into metastability, $K2$ is the time interval available for resolving the metastability, and τ is a device dependent time constant. Note that the formula assumes that the changes of the asynchronous input is uniformly distributed over the clock period. The formula is equivalent to the one presented in [27]. It has not been possible to find information targeting the Virtex-5 FPGA, but it is expected that due to the newer process used the MTBF is further improved.

In the implementation of the synchronizer the two flip-flops should be placed in the same slice component using the `rloc` constraint to minimize the routing delay between them. Details on the use of `rloc` is found in section 2.6.3. The implementation is found in appendix A.5.1.4 (p. 131).

2.6 Controlling Timing

Controlling timing is vital for any digital design. In asynchronous designs the delay matching process is highly dependant of the ability to control path delays in the design.

In section 2.6.1 the predictability of the delay elements is investigated through a series of simulation experiments. In the Xilinx design flow the preferred way to control timing is by assigning timing constraints to the design. The ability to use these timing constraints on asynchronous designs are explained in section

2.6.2. Another method which can ease the delay matching process is the ability to create design macros with repeatable timing metrics. This method is called relationally placed macros. Some problems have been encountered for creating relationally placed macros of asynchronous components. Section 2.6.3 explains this.

2.6.1 Delay Element Experiments

The delay element presented in section 2.5.3 does not give fixed delay lengths for a given size. Even though the delay through a LUT is fixed for all LUTs on the FPGA, variations in the wire routing will lead to variations in the delay produced by the delay element. In this section a number of experiments based on post place and route simulations of the delay element will be presented.

The purpose of the experiments is to document a number points:

- How large is the delay of a delay element of a given size.
- How predictable is the delay of a delay element, i.e. how large are the fluctuations of the produced delay of delay elements with equal sizes.
- How the use of placement constraints affects the predictability.
- If changing the size of a delay element will affect the timing of the datapath, such that the delay to be matched will change.

To investigate if the context in which a delay element is used affects the predictability, the delay element simulations are performed in two scenarios:

- Delay elements alone.
- Delay elements instantiated in a larger design.

By simulating the delay elements in a larger design the fluctuations of the delay of the datapath can be measured.

For the simulations where the delay elements is instantiated in a larger design, the measurements are performed on the delay elements in a FIFO stage of the NoC router presented in section 4.2. The FIFO stage is connected to an input port of the router and the depth of the FIFO is one. No IO buffers are inserted when the design is implemented. A simulation module is used to send data

into the FIFO. Only measurements on the rising edge of the request signals are performed.

After the delay simulations was performed an error was discovered in the design of the FIFO stage.[2] Therefore, the FIFO stage presented in section 4.2 differs from the one used for the delay simulations. This does not affect the conclusions about the delay simulations, since the delay observations are general for any circuit.

The FIFO stage includes three delay elements; one for each of the three request signals. Figure 2.16 shows the section of the FIFO stage used in the simulations.

In the rest of this section the following results will be presented:

- The ratio between gate delays and wire delays in the delay element.
- Comparison of the delay produced by a placement constrained delay element and an unconstrained delay element when simulated alone.
- The same comparison but with the delay elements instantiated in a NoC router.
- Correlation between the size of the delay elements and the delay to be matched in the datapath. Changing the size of a delay element affects the overall placement of the design, resulting in variations in the delay to be matched.

When the delay element is simulated alone, there is no wire delay on the input signal, because it is the only component in the design. For the simulations of the FIFO stage the delays are measured from the output of the C-elements to the output of the delay element, i.e. the wire delay between the C-element and the delay element is included in the measurement. The delay which the delay element must match are measured from the output of the C-element to when data is stable on the output of the latch. In the simulations the size of the delay elements are varied from 2 to 30 LUTs. Since each FIFO stage includes three delay elements, three independent measurements can be made from each simulation. Both post map and post place and route simulations are presented. Because a post map simulation does not include wire delays, the post map delay will be the same for all equal sized delay elements.

The simulation results with delay elements alone are shown in figure 2.17. The *constrained* graph is for the delay element where the LUT placement has been

[2]The latch was wrongly set to be opaque when EN = 0. The latch should be opaque when EN = 1.

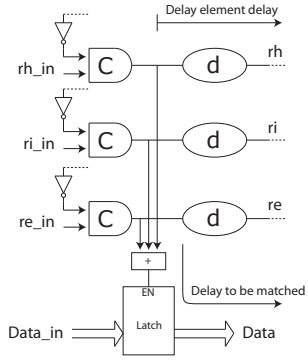


Figure 2.16: Section of the FIFO stage used in the simulations.

constrained as shown in figure 2.13 on page 21. The *unconstrained* graph is a delay element where `rloc` constraints have not been applied. The post map graph is completely linear and satisfies the equation

$$delay = 80 \cdot size$$

which agrees with a LUT delay of 80 ps, as specified in the data sheet. Using linear regression to approximate an equation for the post place and route delays in figure 2.17 (forced through (0,0)) gives

$$delay_{unconstrained} = 352 \cdot size$$
$$delay_{constrained} = 478 \cdot size$$

The gate delay only constitutes from 18% to 23% of the total delay giving approximately a 1:5 ratio between gate and wire delays. In the Xilinx Constraints Guide [29] it is stated that the routing delay typically accounts for 45% to 65% of the total path delay for a combinatorial circuit. So the contribution of the routing delay is larger than expected. Constraining the placement of the delay LUTs results in an average increase in the resulting delay of approximately 35% The predictability of the unconstrained delay element is quite good, with only small fluctuations in the delay. The constrained delay element is even better with almost no fluctuations. The small fluctuations for the constrained delay element can be explained by the fact, that even if the LUTs in the delay element are constrained to a specific slice, the internal placement within the slice can still vary, and also the chosen routing between slices can deviate from one another. The conclusion of the simulations of the delay elements alone is that the predictability is improved for the placement constrained delay elements compared with the unconstrained delay elements but the unconstrained delay elements still produces fairly predictable delays. The constrained delay elements

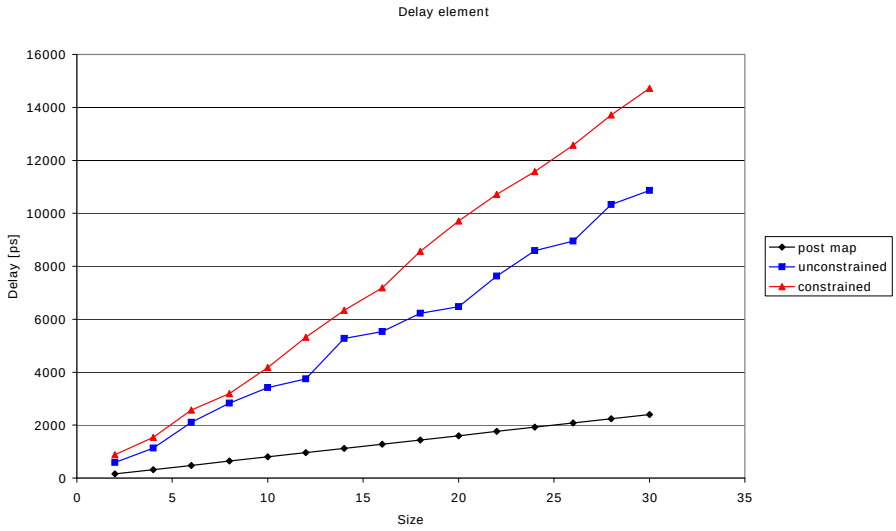
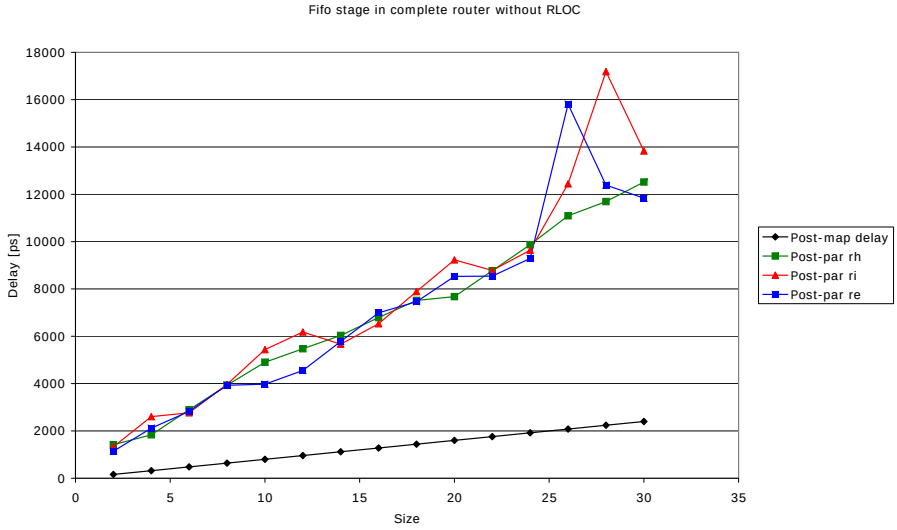


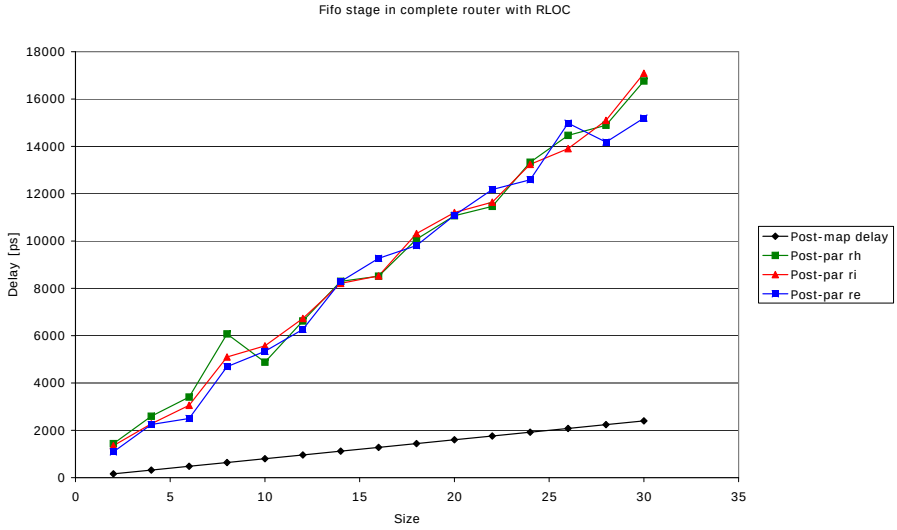
Figure 2.17: Simulations of a single delay element, with and without placement constraints.

produces larger delays for the same size, due to the longer routing caused by the placement.

Figure 2.18 shows the simulation results for the delay elements in the FIFO stage. A stage has 3 request signals: *rh*, *ri*, and *re*. Figure 2.18(a) shows the simulations with the unconstrained delay element and figure 2.18(b) shows the simulations for the constrained delay element. Comparing the unconstrained delay element when it is inserted in a larger design and when it is simulated alone shows comparable predictability for small sizes. For larger sizes significant delay fluctuations are observed. An increase in the size of 2 results in a single case in an additional delay of more than 6 ns. For the constrained delay element the produced delays are free from such large fluctuations. Both the unconstrained and the constrained delay element produces larger delays when inserted in a larger design compared with the single case. The reason for this is the extra wire delay from the output of the C-element to he input to the delay element. Variations of this wire delay can also explain the decreased predictability of the constrained delay element. Constraining the placement of the delay elements increases the predictability of the delay when the delay element is used in a larger design. It is expected that the fluctuations of the unconstrained delay element will be even more noticeable for larger designs with a higher LUT utilization ratio.



(a)



(b)

Figure 2.18: Delay simulations of a FIFO stage. (a) Using unconstrained delay elements. (b) Using constrained delay elements. (a)

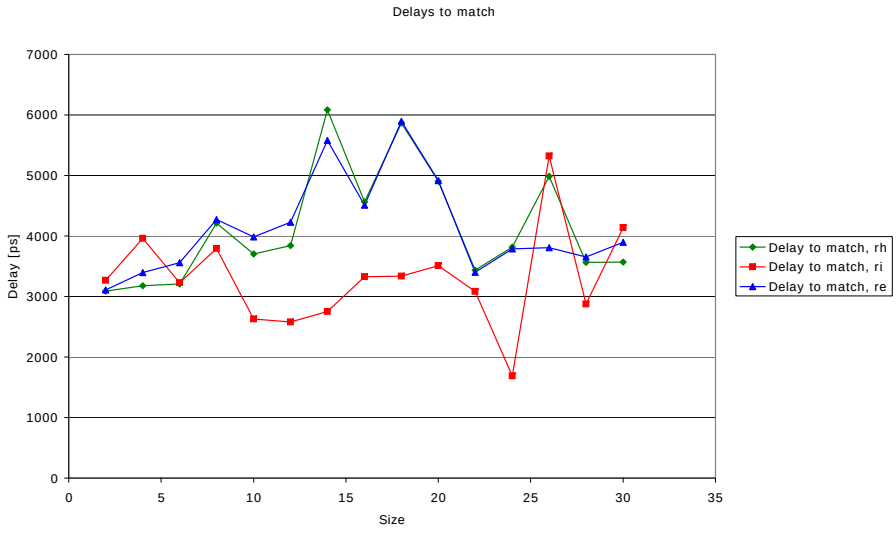


Figure 2.19: Delays in the datapath to be matched.

When performing delay matching of a circuit, changing the size of a delay element will affect the delay that the delay element should match. In fact, even a small change in the design will affect where logic is placed thus altering the routing and thereby changing the timing parameters. To investigate how significant this effect is the size of the delay element versus the delay to be matched in the datapath has been measured. For the simulations the same setup as in figure 2.16 has been used with a complete router design. The measurements are shown in figure 2.19. The x-axis is the size of the delay elements and the y-axis is the time interval from when the request signal is asserted to the output of the latch is stable. The graphs show fluctuations in the delay to be matched of more than 3 ns. This indicates that extra overhead is needed when a circuit is delay matched to account for delay fluctuations in the datapath.

2.6.2 Timing Constraints

In the Xilinx design flow the preferred way to control timing is by assigning timing constraints to the design. This section will describe the timing constraints that are available to control the timing of a design.

The guidelines for assigning timing constraints provided by Xilinx are found in

the Xilinx Constraints Guide [29]. Two groups of timing constraints exists:

Global timing constraints affects all paths in the clock domain. Global timing constraints are used to specify global constraints for clock signals, input/output pads, and combinatorial pin-to-pin paths. They are most commonly used on clock signals.

Specific timing constraints are assigned to a specific path in the design. A specific timing constraint can either be a static path constraint or a multi-cycle path constraint. A multi-cycle path constraint is used when the timing of the path between two registers must be constrained to a multiple of the register clock. A static constraint is assigned to a pad-to-pad path without registers.

All timing constraints are assigned in the UCF file and is applied after synthesis.

To constrain a clock net it must be assigned a name using the `tnm_net` constraint and the desired clock period are assigned to the clock net using the `timespec period` constraint. The design tool will try to optimize the datapath to meet the timing constraint applied to the clock net. If there is not specified any global clock constraints the design tool will identify possible internal clock signals in the design and perform optimizations according to these local clocks. This is referred to as *Performance Evaluation mode* by Xilinx. Performance Evaluation mode is only used when *Timing Driven Packing and Placement* is enabled in the mapper. Timing Driven Packing and Placement is one of the phases of the Xilinx mapping process. For older platforms, than the Virtex-5, timing driven packing and placement was optional, but for the the Virtex-5 it is a required step of the mapping process [32]. In an asynchronous-only design there will typically not be any global clock constraints. Therefore the designer should be aware of the optimizations performed when Performance Evaluation mode is active.

The static path constraints are the only constraints that are not related to a clock, therefore they are the only timing constraints applicable to asynchronous components. When assigning a static path constraint the pad-to-pad delay must be constraint to an absolute time period, e.g. 10 ns. Because timing constraints are assigned to the design after synthesis, the process of assigning constraints to all instances of a component can be cumbersome since all the pin-names must be identified in the post-synthesis net-list.

Static path constraints could be used in the delay matching process. The combinatorial delay experienced by the data signals could be constrained to a reasonable time period. The delay element should then be dimensioned according

to the constrained delay. The problem with this approach is to determine how large the constrained delay should be. It will be hard to avoid a large overhead of the constraint delay, and as a result wasting area and degrading performance due to oversized delay elements. To avoid over-constraining the delay a cumbersome iterative process of design implementation, delay constraining, re-implementation, and delay re-constraining must be applied. This must be done individually for all constrained paths in the design. Nonetheless they use this approach in Aspida [13]. This is manageable because the Aspida design only contains five delay elements and a well-defined datapath with a priori knowledge of the combinatorial delay from the synchronous implementation. In the MPSoC system presented in chapter 7 the number of delay elements exceeds 200. Therefore this approach has been abandoned.

The overall conclusion is that the available timing constraints are not very well suited to control the timing of large asynchronous systems. Due to the manual process of assigning the timing constraints the process becomes too cumbersome, unless the number of constrained paths in the design is very small.

2.6.3 Relationally Placed Macros

For timing critical designs Xilinx provides a method for locking the internal placement of a subcomponent of a design. This method allows the designer to create a *relationally placed macro* (RPM) that can be instantiated in another design with repeatable performance and timing properties. An RPM is a collection of FPGA primitives grouped together in a set in which the placement of each primitive is relationally constraint. This allows the placer to move the macro freely around on the chip area without touching the internal placement.

The relational placement of the primitives is defined using the placement constraint `rloc`. `rloc` is used to assign a primitive to a slice using slice coordinates, e.g. "X0Y0". The slice coordinates was described in section 2.4 (p. 9). If another primitive is assigned to the slice "X1Y0", the two primitives will always be placed in slices next to each other column wise, however nothing is specified about their absolute placement. A guide describing how to create an RPM manually is found in an article from the TechXclusive Xilinx magazine [9] and details about the `rloc` constraint is found in the Xilinx Constraints Guide [29].

RPMs can be created using two different approaches:

- By manually assign `rloc` constraints to FPGA primitives in the design.
- Using Floorplanner to create an RPM from a place and routed design.

The manual assignment of `rloc` constraints is done in the HDL code. A major drawback of this approach is that it can only be used for FPGA primitives directly instantiated in the design. `rloc` cannot be applied in HDL to primitives inferred by the design tools. Obviously this approach is only useful for very small macros. In this project this approach is used in the delay elements in section 2.5.3, in the mutex in section 2.5.2, and in Petrify circuits in section 2.7.2.

The Xilinx Floorplanner tool is able to create an RPM macro based on a placed and routed design. After place and route the design is loaded into Floorplanner, which extracts the relative placement of all primitives as `rloc` constraints and writes them to the UCF file. The netlist and UCF file is then combined to a macro file, which can be instantiated in another design as a black box macro. Detailed information about the RPM creation process can be found in the Xilinx Application note in [31] and in the Floorplanner documentation [30].

When designing an asynchronous system it will be highly desirable to be able to delay match small subcomponents individually and then create an RPM macro component with locked placement. When connecting several RPM macros only the routing between the macros will be able to inflict incorrect timing. To create an RPM of a subcomponent the Floorplanner approach must be used, unless the subcomponent solely consists of instantiated FPGA primitives. Unfortunately it has not been possible to successfully create an RPM macro of an asynchronous design using Floorplanner. In the following the problems encountered will be explained.

When Floorplanner is used to extract the relative placement of the design primitives it does not include all primitives present in the design. Some primitives are present in the Floorplanner design hierarchy but are unplaced. Other primitives are not even present in the Floorplanner design hierarchy even though they are present when the design is loaded into *FPGA Editor*. It has been determined that all problematic primitives are LUTs which is marked as *"route throughs"*. A LUT is used as a "route through"-LUT to let a signal get access to slice resources that is only accessible through a LUT. This situation may arise if the internal slice signal dedicated to bypass the LUT are already used by other logic. Because a "route through" does not perform any function in the design but is solely used as a routing resource, it may be the reason that it is not included in the RPM. However, it has not been possible to find any Xilinx documentation to support this theory. Consequently all C-elements are marked as "route throughs" LUTs and thereby not included in the RPM macro. The marking of a C-element as a "route through" does not really make any sense. As C-elements are a vital part of any asynchronous circuit it is crucial to include them in the RPM. Also simple mux'es and demux'es have suffered from the same problem. Even a strictly combinatorial demux circuit is found to give trouble. It has been tried to rewrite the HDL code to see if that could solve the

problem, without any success. A description of the problem have been posted to the Xilinx user forums and the internet newsgroup `comp.arch.fpga` but no replies have been received. The forum post is included in appendix A.4. Since also non-asynchronous circuits suffer from problems it is suspected to be caused by a bug in the Floorplanner software. It should be noted that RPMs can successfully be created from other combinatorial circuits using the Floorplanner tool.

The inability to create RPMs of asynchronous components has the consequence that a larger margin must be included in the matched delay, however it has not proved to be a major issue as long as performance does not have high priority.

2.7 Design Flow

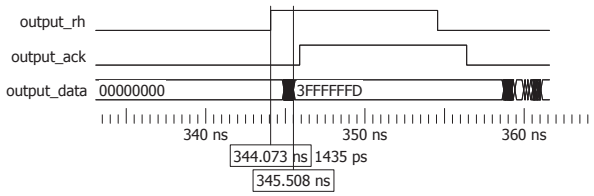
This sections aims at describing the design flow for implementing asynchronous circuits using the Xilinx tools. On the basis of the results from the experiments on the delay element presented in section 2.6.1 a guideline for delay matching is presented in section 2.7.1. The design flow for implementing Petrify circuits is presented in 2.7.2. In section 2.7.3 various settings and constraints used for the Xilinx tools are described.

2.7.1 Delay Matching Guidelines

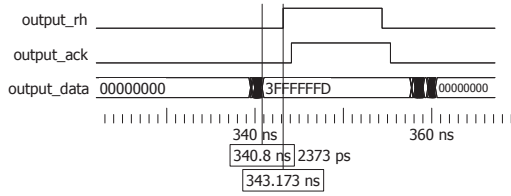
This section describes the work flow for performing delay matching of a circuit. The output port component from the router design (section 4.2.2) is used as an example.

Wire and gate delays will introduce different propagation delays for the handshake signals and for the data signals. The 4-phase bundled data protocol requires that the data are valid before the request signal is asserted. If the request signals have a smaller path delay than the data signal (which they will have in most cases) the request signals must be delayed to obey the handshake protocol. If not, the receiver may latch invalid data.

When measuring the required delay one should make sure that all data signals make a transition, because all the data signals will have different propagation delays. In the example below it has been aimed at that all data signals makes a $0 \rightarrow 1$ transition, but to assure that the packet follows the correct path through the router this has not been possible for all data signals. Figure 2.20(a) shows



(a) before delay matching



(b) after delay matching

Figure 2.20: Modelsim print of a request signal in an router output port before and after delay matching.

a Modelsim print of a handshake transaction in an output port: *output_rh* is the request output and *output_data* is the output of the data latch. A curser marks the time when the request signal is asserted and another curser marks when the output of the data latch is stable. The figure shows a difference of 1.4 ns. The chart in figure 2.18 (b) (p. 28) is used to estimate the size of the delay element. According to the delay chart a size of 3 should give a delay of about 2 ns. To allow for the delay fluctuations mentioned in section 2.6.1 an extra delay should be inserted. The experience from this project is that in general a delay overhead of about 2-3 ns is sufficient to cover the delay fluctuations. Thus, the target is a delay of about 4 ns. According to the delay chart a size of 8 should give a delay in the target range. After the insertion of a delay element with size 8, the design is synthesized and implemented again. A Modelsim print of the handshake transaction after insertion of the delay element is shown in figure 2.20(b). After insertion of the delay element the delay overhead is 2.4 ns, which lies in the target range. It should be noted that a delay overhead of 2-3 ns is a quite conservative estimate. In many cases a smaller overhead will be sufficient.

In a larger design with many instances of the same component it is not feasible to manually check if each and every delay element is sufficiently large. By following the handshake protocol it is guaranteed that the correct data is latched, but even if the handshake protocol is not completely obeyed data might be correctly

latched anyway. In other words, the primary goal is not to assure that the handshake protocol is strictly followed under all circumstances but to assure that correct data is latched in all cases. The Modelsim simulation tool will issue warnings if a latch experience setup or hold time violations. If the simulation of the complete system does not result in any warnings it indicates that the delay matching is sufficient.

2.7.2 Petrify Circuits

This section presents the design flow for implementing control circuits synthesized by Petrify [6].

Petrify is a tool which synthesizes speed-independent control circuits specified by State-Transition Graphs (STGs). An STG is a way to specify a timing diagram in a formal way and is based on Petri nets. The tool *Visual STG Lab* (VSTGL) is a visual tool for creating and simulating STGs and it has been used in this project for the creation of STGs. The input to Petrify is an STG and the output is a set of boolean equation which implements the circuit. Petrify automatically solves Complete State Coding (CSC) violations by inserting extra state variables, however the designer should try to limit the amount of needed CSC state variables to as few as possible. The amount of needed CSC state variables can be reduced by redesigning the STG specification.

The general process of implementing a circuit specified by an STG and synthesized by Petrify is described in chapter 6 in [24]. The process of mapping a set of petrify equations onto an FPGA is described in [26]. Two different methods is presented in [26]:

- Complex gates.
- Generalized C-elements.

When the target is a complex gate implementation Petrify generates equations such that each non-input signal is implemented by a single complex gate. The `-cg` option is used to instruct Petrify to target a complex gate implementation. A complex gate must be implemented in a single LUT element, hence the number of inputs to a complex gate is limited by the number of inputs available on a LUT. The state-holding capabilities of the complex gate is implemented as a feedback input, in the same way as in the implementation of the C-element in section 2.5.1. The reset signal can be implemented using an internal MUX in the slice such that it does not occupy an input on the LUT. In [26] an FPGA

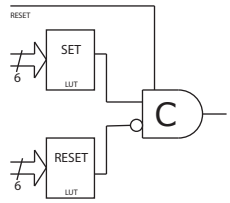


Figure 2.21: Implementation of a Petrify circuit using a C-element.

with 4-input LUTs is used, thereby leaving 3 inputs free. With the 6-input LUTs available in the Virtex-5 FPGA a 5-input complex gates is the maximally possible to implement. If more inputs is needed Petrify must be used to do speed-independent preserving decomposition else the circuit will not be hazard free.

A solution based on generalized C-elements uses a state-holding element. The state-holding element can be either a set-reset latch or a C-element. Petrify generates equations implementing the set and reset functions for the state-holding element. When a SR latch is used the set and reset functions are wired to the set and reset input of the SR latch respectively. When a C-element is used the set function is wired to one input and the complemented reset function is wired to the other input. To be able to control the initial state of the SR latch or the C-element a reset signal must be used. For the SR latch implementation an internal mux can be used to save an input of the LUTs implementing the set/reset functions as with the complex gate. The C-elements already have a separate reset input. The set/reset functions can have up to six inputs. If a larger number of inputs is needed several LUTs can be combined in a sum-of-products configuration. To assure that the sum-of-product implementation is hazard free, Petrify must be instructed to apply the *monotonic cover constraint*, using the `-gcm` option. With the monotonic cover constraint only one term in the sum-of-products implementation is allowed to be high at a time, thus eliminating the possibilities of static and dynamic hazards. In [26] the generalized C-elements are implemented using SR latches.

The set/reset functions encountered in this project has a maximum of 6 inputs, hence every set/reset functions can be implemented in a single LUT primitive. In the project C-elements is used as the state-holding element for implementing Petrify circuits. The C-element solution is easier to implement than SR latches but it should be noted that a solution with SR latches is a more “correct” solution since it is available as a FPGA primitive. Figure 2.21 shows a generalized C-elements implementation using a C-element.

When implementing the boolean equations representing the set and reset functions it is tempting to let the Xilinx tool do the mapping to LUTs, but this may lead to corruption of the circuit. The synthesizer will try to reduce the logic expressions as much as possible; a task that it is brutally good at. To maintain speed-independence Petrify may insert terms in the boolean expressions which will seem redundant to the synthesizer, consequently they will be optimized away. To circumvent these logic optimizations the designer must do the mapping to LUTs manually, by instantiating the LUT primitives with the desired logic function directly in the HDL code. The implementation of the C-element presented in section 2.5.1 has non-inverted inputs. For the implementation of Petrify circuits a C-element with one inverted input is used. It only differs from the original C-element by a slight change in the init value.

The circuit generated by Petrify assumes a speed-independent delay model. Speed-independence assumes positive, bounded but unknown gate delays and ideal zero-delay wires [24]. Assuming ideal wires is of course not very realistic but the wire delay can in most cases be lumped into the gate delay for the purpose of delay analysis. Problems may arise if an output is used in several inputs. If the fork is non-isochronic, i.e. the end-points of the fork experience different wire delays, the circuit cannot be considered speed-independent. The forks in an FPGA implemented circuit should always be considered as non-isochronic. In most cases a circuit with non-isochronic forks will work as intended, however an unfortunate combination of wire delays where one end of the fork is much slower than the other, may lead to a circuit malfunction. To circumvent this problem the relational placement between all LUTs in the Petrify circuit is locked using the `rloc` placement constraint. This will minimize the possible delay fluctuations between different instantiations of the same Petrify circuit. The strategy used for selecting a relational placement is very simple. Pick an arbitrary placement where the LUTs are placed next to each other. Do a post place and route simulation to verify that the circuit works as intended. If the simulation fails, locate the the faulty signals and replace the affected LUTs. A more analytical approach where the problematic forks is located beforehand could be applied, but it has not been considered to be worth the trouble for the relatively simple Petrify circuits implemented in this project.

For a circuit to work as specified it must be properly initialized. The initialization values for the state-holding elements that is required for correct functionality is listed by Petrify.

To summarize the procedure for implementing Petrify circuits used in the project:

- Draw and simulate the STG in VSTGL.
- Synthesize with Petrigy using the `-gcm` option to use generalized C-elements and apply the monotonic cover constraint.
- Implement all boolean equations in instantiated LUTs.
- Generalized C-elements is implemented using a C-element component.
- Set the initialization values.
- Lock the relational placement of all components using the `rloc` constraint.

2.7.3 Tool Settings and Constraints

The Xilinx design tools have a large amount of settings and constraints to control the synthesis and implementation processes. In this section the use of some of these settings are explained:

- Optimization settings for the synthesis and mapping process.
- The `optimize` constraint.
- The `keep` constraint.
- The `tig` constraint.
- The `keep hierarchy` setting.
- The use of clock buffers.

In general we want the design tool to perform as few optimizations on the asynchronous components as possible, because the tool will not "understand" the asynchronous circuits. For Petrify circuits (section 2.7.2) it is absolutely crucial that no optimizations are performed at all, thus mapping the design to LUTs must be done manually. For other asynchronous components the optimizations should be kept to a minimum. The process of implementing a circuit is done in roughly three steps: synthesis, mapping, and place and route. Design optimizations, that may alter the logical function of the design, are performed during the synthesis and the mapping processes.

Goal	LUTs	HS. Cycle
Area	2050	42 ns
Speed	2210	38 ns
Difference	7.8%	5.3%

Table 2.1: Effect of synthesis optimization settings.

It is not possible to turn of the logic optimization in the synthesizer. The optimization goal can be set to either *speed* or *area* and the effort level to *normal* or *high*. Apart from Petrify circuits there has not been observed any incidents where synthesis optimization has caused failures. The optimizations settings are global for the design, thus in a mixed design with both synchronous and asynchronous components all components are affected by the setting. Table 2.1 show difference in area and performance for a router using a different optimization goals. The focus of this project has not been performance, so the synthesis optimization goal has been set to area in all designs.

In the mapping process optimizations are performed during the *cover phase* where logic are assigned to LUTs. The optimization goal can be set to: area, speed, balanced, or off. The optimization setting can be set either globally or individually for a component. The global setting is set in the mapping properties. To use a different optimization setting for an individual component the `optimize` constraint is used on the VHDL entity. In an asynchronous-only design the mapping optimization goal should be set to `off` globally, and in a mixed design the `optimize` constraint should be used to turn off optimizations for the asynchronous modules only. Optionally the mapper can perform post-placement logic optimizations to improve timing using the `logic_opt` switch. This is set to `off` by default, and should be left like that.

Even with the mapping cover setting set to off, some optimizations are still performed during the mapping process. An example of this is the delay element, where the `keep` attribute must be assigned to the signals connecting the LUTs, or else the mapper will optimize the delay element into a single LUT. When the `keep` constraint is attached to a signal it will prevent that the signal is absorbed into a logic block caused by optimizations, consequently the signal is kept in the final net-list. To minimize the possibility that the mapper removes important logic, it is advisable to apply the `keep` constraint to all signals within an asynchronous component, even though it is not necessary in most cases.

In a GALS design with both synchronous and asynchronous components, the asynchronous components must be excluded from the timing analysis performed by the design tools. If the asynchronous components are not excluded, the combinatorial delay of the asynchronous components will be included in the

maximum path delay used to determine the maximum clock frequency. This situation has similarities with a multi-clock synchronous design where signals may cross clock domains. The `tig` (timing ignore) constraint is used in these situations to exclude a static path from the timing analysis. The `tig` constraints tells the tool to ignore all paths fanning forward from the `tig`-marked net to be ignored during timing analysis. `tig` should be assigned to all nets going into the asynchronous design. The `tig` is assigned to nets in the UCF file. If the `tig` constraint is not used it will be very hard for the tool to meet the clock constraints because the combinatorial delay of the asynchronous components is included in the critical path. For larger designs the mapping process will even fail completely. In an asynchronous-only design there are no clock constraints to meet, however the `tig` constraint still affects the run-time of the mapping process. Due to the Performance Evaluation mode discussed in section 2.6.2 the tool will try to perform timing optimization based on the local clocks it finds in the design. In Aspida [13] the `tig` constraint is only used on the delay elements to avoid timing optimizations of the delay elements.

The design tool automatically infers clock buffers on signals it believes to be clock signals. A clock buffer causes the signal to be routed on special low skew routing resources. Within asynchronous components the tool will find clock signals and if there are unused clock buffers, it will infer it on the signal. To avoid this from happening the synthesis property `Number of Clock Buffers` should be set to 0. If the design contains any clock signals clock buffers must be inserted manually.

In the synthesis properties it can be chosen if the design hierarchy should be kept or the design should be flattened. If the design hierarchy is flattened optimizations are performed across hierarchical components. Therefor a flattened design typically uses less resources. A disadvantage is that it makes it a lot harder to locate signals in the post-synthesis simulation models. For the purpose of asynchronous design where post place and route simulations are used extensively it is a big advantage to keep the hierarchy for this reason only.

Chapter 3

Networks-on-Chip

This chapter will present the basic theory behind NoC design. The first section 3.1 will give a brief introduction to the general NoC design paradigm. In section 3.2 the basic concepts of NoC design will be presented. The last section 3.3 will present the previous work in the field of NoC design that have been used for this project.

3.1 Introduction to Networks-on-Chip

NoC is an emerging design paradigm for designing the interconnect for large SoCs. More common SoC interconnects such as busses and point-to-point links scales poorly when the number of IP cores in the system is increased. The NoC design paradigm tries to handle the scaling problem of the bus and point-to-point interconnects.

In large SoC systems a Globally Asynchronous Locally Synchronous (GALS) design approach is advantageous, due to the difficulties with clock distribution for large SoC systems. In a GALS system each core operates in their own local clock domain. Thus, the need for a global clock is eliminated. Due to the lack of a global clock is seems very intuitively to have an asynchronous interconnect in

a GALS system. Therefore an asynchronous NoC matches the design challenges for a GALS based SoC well.

NoC design shares many similarities with the design of parallel computer networks. Therefore a large amount of the research carried out in this field is also applicable for NoC design.

3.2 Basic Concepts

This section presents the basic concepts of the NoC paradigm. The source of the theory presented in this section is from the article *A Survey of Research and Practices of Network-on-Chip* (part of the MANGO PhD thesis [5]) and chapter 10 from the book *Parallel Computer Architecture – A Hardware/Software Approach* [7].

A NoC consists of four fundamental components: *IP Cores*, *Network Adapters*, *Routers*, and *Links*. A description of each of the NoC components is found below.

IP Cores The purpose of the NoC is to let the cores in the system communicate with each other efficiently. Thus, the cores are not an actual part of the NoC. A core can initiate requests (a master core), or respond to requests (a slave core) or both. Typical examples of master and slave cores are CPUs and memories respectively.

Network Adapters (NAs) provides an interface for the core to communicate with the NoC. Typically the cores uses a memory-mapped interface while the network is based on message-passing. The NA must translate between the two types of interfaces. The NA must also handle synchronization issues between the cores and the network.

Routers are connected to each other in a structural manner to create a network for the cores to communicate on. The routers route incoming packets to the desired destination based on a routing algorithm.

Links are used to to connect the routers to each other. The links consists of control wires and data wires.

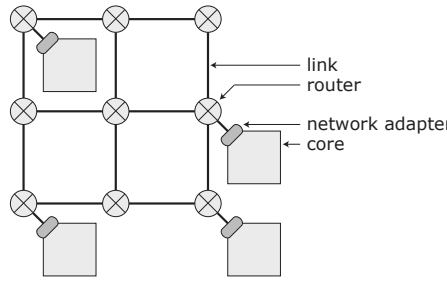


Figure 3.1: An example of a NoC connected in a 3-by-3 mesh topology.

In figure 3.1 an example of a NoC with four cores connected in a 3-by-3 mesh topology is shown. The main properties of a NoC is the choice of *topology*, *routing algorithm*, *switching strategy*, and *flow control*.

- **The topology** determines how the network components are physically connected to each other.
- **The routing algorithm** determines the route that messages follows through the network.
- **The switching strategy** determines how messages traverses the route.
- **Flow control** determines when messages traverses the route.

The following sections will describe these properties.

3.2.1 Topologies

The topology describes how the routers are connected to each other. The choice of topology affects many aspects of the NoC, e.g. area, performance, and power. Many different topologies exists, e.g. meshes, tori, cubes, butterflies, and trees. Topologies can be either regular or irregular, where irregular topologies can be created by mixing different types of regular topologies. A *direct* network have cores connected to all router nodes, while an *indirect* network only have cores connected to a subset of the router nodes. The network in figure 3.1 is an example of an indirect network.

In general the choice of topology is a tradeoff between the number of links between the routers vs. cost. In a topology with many links the average routing

distance between two cores will be small and consequently the latency through the network will be low. Most NoCs uses topologies with relatively few links that is easy to lay out on a 2-d surface. Meshes, tori, and trees have this property and are therefore popular for NoCs. In the following the mesh and torus topologies will be described.

Meshes and tori (and cubes) are known together as k-ary d-cubes, where k is the degree of each dimension and d is the number of dimensions. The mesh in figure 3.1 is a 3-ary 2-cube mesh topology. A torus is different from a mesh by that it connects opposite edges. Therefore tori often uses unidirectional links, while meshes uses bidirectional links. Tori does not perform well in networks with with a high amount of local traffic due to the unidirectional links.

3.2.2 Switching strategies

A network can be either packet switched or circuit switched. In a circuit switched network the route from source to destination is setup before the message is transmitted on the network and is not taken down before the transmission has finished. In a packet switched network the message is split up in a number of packets that are individually routed from source to destination. Each packet contains routing and sequence information. The majority of the developed NoCs utilizes packet switching [5]. The minimum amount of data that can be sent between two nodes in the network is called a flow control unit (flit). A packet consists of several flits which are transmitted in series. For packet switched networks different packet switching strategies exists: store-and-forward, wormhole, and virtual cut-through. The three switching strategies are explained below.

Store-and-forward all flits of a packet are received by the node, before any flits are forwarded to the next node. If the next node does not have sufficient buffer space, the packet is stalled.

Wormhole when a node receives a flit of packet, the flit is forwarded to the next node as soon as possible. The tail of the packet is left behind in the nodes along the route. If the header flit is blocked, all links spanned by the packet will be blocked.

Virtual cut-through is a compromise between store-and-forward and wormhole. Forwarding of flits works in the same way as for the wormhole strategy, however the header flit is only forwarded to the next node if it has sufficient buffer space to store the complete packet.

Protocol	Per router cost:		Stalling
	Latency	Buffering	
Store-and-forward	packet	packet	at two nodes and the link between them
Wormhole	header	header	at all nodes and links spanned by the packet
Virtual cut-through	header	packet	at the local node

Table 3.1: Costing and stalling for the different switching strategies, from [5].

The wormhole strategy has lower latency and lower buffer requirements compared to store-and-forward. The disadvantage is that the possibility of deadlocks in the network increases, because a packet can span several links. Virtual-cut through will benefit from the low latency of the wormhole strategy under normal load. Under high load the virtual-cut through strategy will approach the function of a store-and-forward network, where flits are aggregated in the front node. Thereby the increased possibility of deadlocks from wormhole is removed. However, virtual cut-through still have the large buffer requirements of the store-and-forward strategy. Therefore wormhole is the most popular switching strategy for NoCs. Table 3.1 summarizes the latency penalty and storage cost for each of the discussed switching strategies.

3.2.3 Routing algorithms

Routing algorithms are divided into two categories: deterministic and adaptive. For a deterministic routing algorithm the chosen route is based solely on the source and destination, i.e. there are only one legal path between a pair of nodes.

An adaptive algorithm allows more than one legal path between a pair of nodes and the route is determined on a per-hop basis. The choice of routing path is dynamically determined based on e.g. link congestion. Because each router must be able to dynamically decide the routing direction, the complexity of the router implementation increases. The advantage of an adaptive routing algorithm is that it allows for better link utilization, which can improve performance considerably, especially under high load.

A deterministic routing algorithm can be source-routed or use table-driven routing. With source-routing the source must provide the complete routing path to the destination in the packet header. Source-routing simplifies the process of determining the routing direction within the router. The router can directly determine the routing direction by looking at the packet header. However, if the

packet header has a fixed length, the maximum number of hops in the network is limited by the header length. In table-driven routing each router contains a routing table where it can look up the routing direction based on the destination written in the packet header. Thus, the maximum number of hops is not limited by the header length. The implementation of the routing table will increases the area of the router.

Some regular topologies allows for very simple routing algorithms. For example in a k-ary d-cubes topology, *dimension order routing* can be used. In dimension order routing for a 2-d mesh, the packet is first routed fully in the x-direction and then fully in the y-direction. The packet header contains the remaining distance for each dimension and the routing direction is easily determined from the remaining distance. Before a router forwards the packet it decrements one of the distances in the header, in accordance with the routing direction. In k-ary 2-cubes dimension order routing are also known as *xy-routing*. For a source-routed network the source simply inserts a routing path in the header in accordance with the dimension order routing algorithm.

A routing algorithm can be either *minimal* or *non-minimal*. A minimal routing algorithm always selects the shortest path between source and destination, while a non-minimal algorithm is allowed to select longer routes between source and destination. Dimension order routing is an example of a minimal routing algorithm while most adaptive algorithms is non-minimal due to their dynamic behavior.

An important aspect of a routing algorithm is, whether it is *deadlock free*, for the topology in which it is used. Routing deadlocks may occur in a network when for example several packets are waiting for each other in a cyclic dependency. If this happens, none of the packets are able to make any progress. The process of determining if a routing algorithm is deadlock free for a given topology can be quite cumbersome. In [7] a guide for determining deadlock freedom is presented. The xy-routing algorithm described above has the interesting property that it is proved to be deadlock free for a k-ary 2-cube mesh topology [7]. It is important to note that the algorithm is only proved to be free from routing deadlocks. Deadlocks may occur in higher levels of the system. E.g. if the NAs are not able to consume packets due to dependencies between the cores in the network. Due to the deadlock freedom of xy-routing it is a popular choice for NoCs.

3.2.4 Flow control

Flow control is the mechanisms used to determine when a message moves along its route [7]. Flow control is mainly used to ensure correct operation of the

network, but can also be used to improve utilization of network resources and to provide predictable performance.

Ensuring correct operation of the network is first and foremost about avoiding deadlocks in the network. For more advanced routing algorithms than *xy*-routing, deadlock problems can solved by introducing *Virtual Channels* (VCs). VCs are the primary method used for assuring deadlock freedom in wormhole routed networks. A VC is created by dividing a physical channel into a set of logical separated channels by adding multiple buffers to the physical channel. To resolve deadlocks VCs are used to systematically break cyclic dependencies in the network. VCs can also be used to increase wire utilization, improve performance and to provide Quality-of-Service (QoS).

Two basic types of QoS exists: *best-effort* services (BE) and *guaranteed* services (GS) [5]. In a BE NoC guarantees are only given for correctness and completion of a transaction, while in a GS NoC performance guarantees such as bandwidth and latency guarantees are given. Naturally the complexity of a BE NoC is much less than a GS NoC.

3.3 Previous Work

NoC research is an emerging field within the SoC research area. Different research groups have published articles about the concepts and implementations of NoC systems. Both synchronous and asynchronous NoCs have been developed. For this project three asynchronous NoC implementations have been studied: MANGO [5], QNoC [23], and Chain [4].

MANGO (Message-passing Asynchronous Network-on-Chip providing Guaranteed Services over OCP interfaces) is a NoC developed at IMM, DTU. MANGO is an asynchronous NoC that provides guaranteed services. The cores connects to the NoC using the OCP interface. The NoC includes both a BE network and a GS network. Connection-oriented GS services providing hard bandwidth and latency guarantees are implemented using virtual channels. Packets are wormhole routed in the BE network and source routing is used. In the M.Sc. thesis *OCP Based Adapter for Network-on-Chip* by Rasmus Grøndahl Olsen [17] an improved design of a NA for the MANGO NoC is presented.

QNoC (Quality-Of-Service NoC) is developed at the Israel Institute of Technology. It is an asynchronous multi-service NoC. The QNoC router is composed of multiple connected input and output ports. The routers are connected in a 2D mesh topology. Credit-based flow control is used to enhance throughput.

Four classes of services are provided. The article presents both a single service level router and a multi-service level router. The NoC uses both source and wormhole routing.

Chain (Chip Area Interconnect) is an asynchronous BE NoC developed at the University of Manchester. Chain uses delay-insensitive 1-of-4 for data encoding and a separate wire is used to signal the end of a packet. The packets in the network are source routed.

Chapter 4

Asynchronous Network-on-Chip Design

This chapter will present the design of the asynchronous NoC developed in the project. Section 4.1 will explain the general design decisions for the developed NoC. In section 4.2 the design of the router is presented and the NA design is presented in section 4.3. Finally in section 4.4 a simple traffic generator for the NoC is presented.

4.1 General Network Design

Implementation of asynchronous systems on FPGAs are experimental therefore only a simple BE NoC has been designed. In general the focus has been on simplicity, while performance have had low priority. The available logic resources on the FPGA is limited so keeping the area low have also had priority. The following sections will explain the design decisions made for the general NoC design.

4.1.1 Topology

When choosing the topology a topology that maps well onto the structure of the FPGA should be selected. The complexity of the topology should also be low. A k-ary 2-cube mesh or torus with unidirectional links are good candidates. A k-ary 2-cube mesh topology with bidirectional links is chosen for the NoC. The main reason for this is that it is easier to guarantee deadlock freedom in the network due to the increased number of links compared to a torus. Combined with xy-routing deadlock freedom can be guaranteed without implementing virtual channels. The 2-dimensional structure of the topology maps well onto the structure of the FPGA.

A k-ary 2-cube mesh topology adds the following requirements to the interface of the router: four ports for network connections, one port for a core connection, and ports must be bidirectional.

4.1.2 Routing

When choosing the routing strategy emphasis has been on simplicity and reduced area utilization.

To reduce the buffer requirements wormhole routing is used. Hence, each router is only required to buffer a single flit. Consequently packets are allowed to contain an arbitrary number of flits. Wormhole routing also benefits from reduced packet latency.

To reduce the complexity of the routers a source-routed scheme has been used. Thus the problem of determining the next hop is moved away from the routers to the NAs. With source-routing the routers will support any routing algorithm in which the source is able to determine the complete routing path to the receiver, but to ensure deadlock freedom it is restricted to use the xy-routing algorithm.

The network does not allow a packet to be routed back in the same direction as it come from. For the mesh topology that leaves an incoming packet at the router with four possible routing directions. Therefore only two bits are needed to encode the routing direction for each hop.

The implementation of virtual channels requires added buffer capacity and added control circuitry and thereby increases the complexity of the router. Adding virtual circuits to a BE NoC is done for mainly two reasons: deadlock avoidance and to increase performance. Deadlock freedom is already guaran-

teed by the chosen routing algorithm and performance does not have priority. Therefore virtual channels have not been implemented.

4.1.3 Handshake Channel

The 4-phase bundled data handshake protocol is used. The handshake protocol is extended with two additional request signals which are used for encoding of the flit type. This is explained in detail in the next section. All handshake channels are push channels, hence it is always the sender that initiates the transaction.

4.1.4 Flit format

A packet consists of several flits. To identify the beginning and the end of a packet three types of flits are required:

- A *header flit* indicates the start of a packet and carries the routing information.
- An *end flit* indicates the end of a packet.
- *Intermediate flits* are all the flits in between the header flit and the end flit.

The header flit holds the routing path and the subsequent flits holds the packet data. Each packet consists of exactly one header flit and one end flit. In between zero or more intermediate flits are allowed. Thus a packet can consist of any arbitrary number of flits.

The flit type must be encoded into the flit. The conventional way to encode the flit type is to append the type as extra data bits. This is the approach used in MANGO [5] and QNoC [23]. In MANGO the BE router uses a single end-of-packet bit to indicate the flit type. The header flit is identified as the first flit to arrive after an end-of-packet flit. In QNoC they have three flit types similar to the types explained above and they are encoded in two bits.

In this NoC a different approach is used. In Chain [4] a 1-to-4 encoding of the data is used and a separate wire is used to indicate end-of-packet. In this project a similar approach is used to encode the flit type. Instead of appending the flit type to the flit data, the handshake channel is used to encode the flit

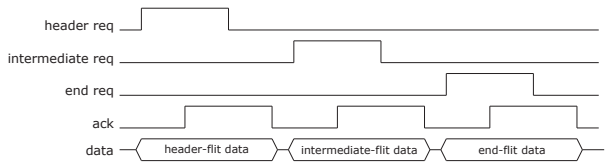


Figure 4.1: The handshake channel used in the NoC.

type. The conventional 4-phase bundled-data handshake channel presented in section 2.2.1 has one request signal and one acknowledge signal. By adding another two request signals, the flit type can be encoded into the handshake channel. Figure 4.1 shows a scenario where a 3-flit packet is transmitted on the handshake channel using this approach. The flit type is identified by one of the three request types: *header request*, *intermediate request*, and *end request*.

By encoding the flit type using request signals the complexity of the routers are further reduced. The control circuits in the router are simplified, since the flit type does not have to be extracted from the data channel. The latency through the router is also reduced because the flit type is known immediately, so it is not necessary to wait for the identification of the flit type. The width of the handshake channel is not affected because the extra request signal should have been routed as data signals anyway. Therefore, no additional signals are needed in the handshake channel. However the complexity of the ordinary handshake components increases due to the additional request signals. Another disadvantage is that three delay elements is needed when a channel is delay matched.

4.1.5 Core Interface

The interface by which the cores connects to the NA is called the Core Interface (CI).

Two different core interfaces have been considered: Open Cores Protocol (OCP) [16] and WISHBONE [28]. The WISHBONE protocol is an open specification provided by OpenCores [19]. OpenCores is an open source community for IP cores. OCP is specified by the OCP International Partnership that has members from a number of large electronics companies. The two specifications are very similar and they both fits the purpose of the project. The OCP protocol has been chosen due to prior use in earlier projects developed at IMM,DTU. This will allow reuse of components developed in the previous projects.

Name	Width	Driver	Function
Clk	1		OCP clock
MCmd	3	master	Transfer command
MAddr	configurable(32)	master	Transfer address
MData	configurable(32)	master	Write data
SCmdAccept	1	slave	slave accepts transfer
SResp	2	slave	Transfer response
SData	configurable(32)	slave	Read data

Table 4.1: Required OCP signals.

For the purpose of this project only a small subset of the features in the OCP specification is needed. Only the features required to connect with the cores used in the multi-processor SoC system presented in chapter 7 is implemented. Thus the NA does not support all the required features to be OCP compliant.

The following OCP commands specified in the OCP specification must be supported by the NA:

Simple write and read transfer Read and write requests sent by the master are accepted by the slave in the same clock cycle. The read response is sent in the immediately following clock cycle.

Write with request handshake The slave is allowed to delay the acceptance of the write requests by an arbitrary number of clock cycles.

Read with request handshake and separate response The slave is allowed to delay the acceptance of the read request by an arbitrary number of clock cycles and the slave is also allowed to delay the read response by an arbitrary number of clock cycles.

The OCP signals required to implement the required commands are listed in table 4.1. The width of the address and data signals are 32 bit because the target CPU has an address and data width of 32.

4.1.6 Packet Format

To support the required OCP commands, three packet types are

- Write Request packet.
- Read Request packet.
- Read Response packet.

The write request and read request packets are sent from the master cores to the slave cores and the read response packet is sent from the slave cores to the master cores. The flit size is set to 32 bits since it is the width of the address and data signals of the OCP interface. If a smaller flit size is selected more flits is required to send a packet and therefore the packet latency will be increased. However, the routing complexity of the routers will be reduced due to the narrower data signals. The routing information for the next hop is stored in two bits as the two MSBs of the header flit. Table 4.2 shows the packet formats for the three packet types.

Flit Name	Flit Type	Width	Description
header flit	header	32	Routing information
control flit	immediate	7	MCmd and MByteEn
addr flit	immediate	32	MAddr
data flit	end	32	MData

(a) Write request packet

Flit Name	Flit Type	Width	Description
header flit	header	32	Routing information
control flit	immediate	7	MCmd and MByteEn
addr flit	immediate	32	MAddr
data flit	end	32	Return routing path

(b) Read request packet

Flit Name	Flit Type	Width	Description
header flit	header	32	Routing information
control flit	immediate	2	SResp
data flit	end	32	SData

(c) Read response packet

Table 4.2: Specification of the packet formats. (a) is the write request packet, (b) is the read request packet, and (c) is the read response packet.

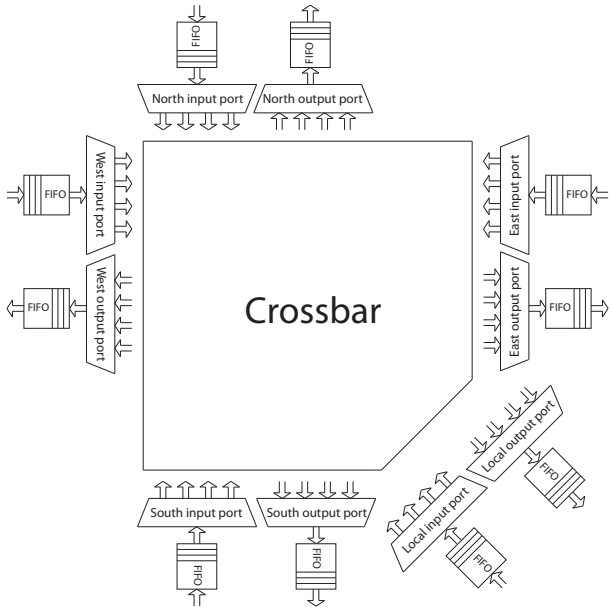


Figure 4.2: The router.

4.2 Router Design

The design of the router is highly dependent on the network in which is it used. The choice of network topology, switching mechanisms, routing algorithm, and flow control mechanisms all influences the requirements to the router.

The router must support a k-ary 2-cube mesh topology, consequently it must provide five port: four for connecting with other routers and one for connecting an IP core. The links are bidirectional so each port consists of an input port and an output port. The router has a non-blocking crossbar, i.e. every input port can be connected to any output port in any permutation simultaneously. FIFO buffers are inserted at the interface of both input ports and output ports. The depth of the FIFO buffers are configurable. Figure 4.2 shows a diagram of the router. In the following sections the design of the input port, the output port, and the FIFO buffers are presented.

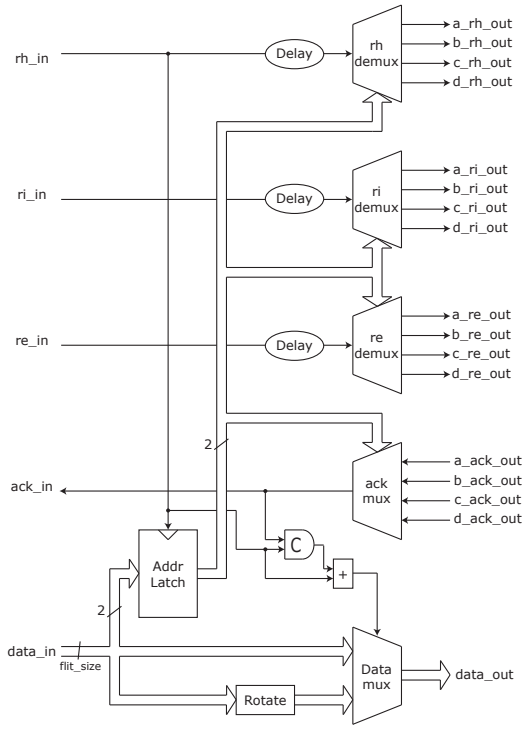


Figure 4.3: The input port.

4.2.1 Input Port

The purpose of the input port is to route the packet to the correct output port. The input port has one input handshake channel and four output handshake channels. The routing direction is controlled by a set of multiplexers. A diagram of the input port design is shown in figure 4.3. The header, intermediate, and end requests are denoted *rh*, *ri*, and *re* respectively.

The first flit arriving is the header flit which contains the routing direction. The routing direction is stored in the two MSBs of the header flit. The two routing direction bits are latched and used as control inputs to the output multiplexers. The routing direction must be locked to the same destination for all subsequent flits belonging to the same packet. Therefore the latch is controlled by the header request signal such that the latch is transparent when *rh* is high. To assure that setup and hold times are not violated, the data validity scheme for the input channel must be broad.

Depending on the flit type the data signal must be treated differently. If it is an intermediate or an end flit, the data should be passed through untouched. If it is a header flit, the data must be rotated two bits. The rotation is done by a rotate component and a multiplexer is used to switch between the two data signals. To maintain a broad data validity scheme the data multiplexer is controlled by a small control circuit consisting of a C-element and an OR gate, with the header request signal and the acknowledge signal as inputs. In case of a header flit the multiplexer selects the rotated data signal and keeps the selection for the complete handshake cycle.

Delay elements must be inserted on all three request channels. The delay elements on the intermediate and the end request signal must match the delay of the data multiplexer subtracted by the delay of the request de-multiplexer. Therefore the matched delay is quite small. The delay element on the header request signal must delay the request signal, until the control signal for the request de-multiplexer is stable. If the delay is not sufficiently large a glitch may appear on one of the header request output signals. The delay must also be long enough for the rotation and multiplexing of the data signal. Consequently the delay element for the header request signal must be larger than the other two.

4.2.2 Output Port

The output port has four input channels and one output channel. The output port must arbitrate between contending inputs, such that only one input channel is granted access to the output channel at a time. Once an input channel has gained access, it must keep exclusive access until the the complete packet has been transmitted. The completion of a packet is indicated by the receival of an end flit. A merge component (see section 4.2.3) is used to merge the four input channel onto the output channel. A diagram of the output port is shown in figure 4.4.

The arbitration is handled by a set of access control circuits and a 4-input mutex component (see section 4.2.4). Because the flit types are encoded using request signals the arbitration between contending inputs can be done in a simple way. Each input channel has associated an access control circuit. When an access control circuit receives a header request, it will request the mutex for access to the output port. When access is granted by the mutex the header flit is passed through to the output. The mutex is not released before an end flit is received. Other contending inputs will wait silently, with a asserted header request signal, for the mutex to grant them access to the output channel.

The access control circuit is specified by the STG showed in figure 4.5. The

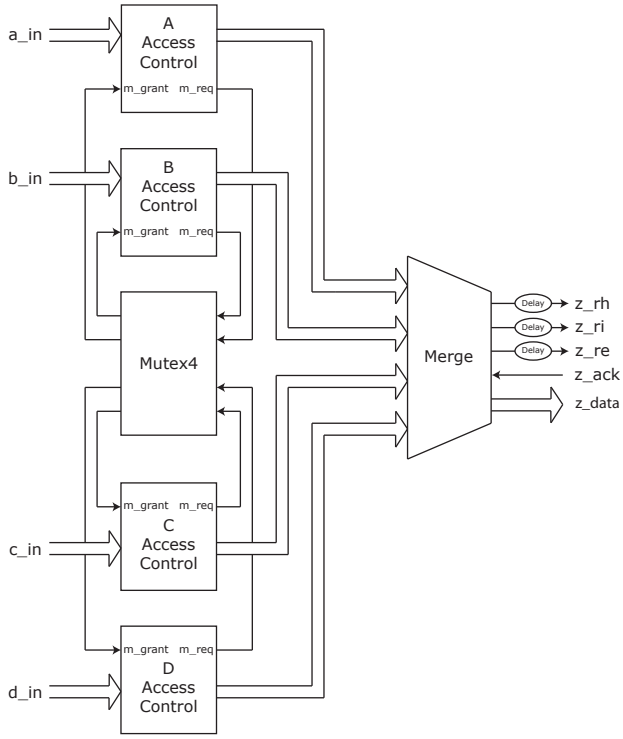


Figure 4.4: The output port.

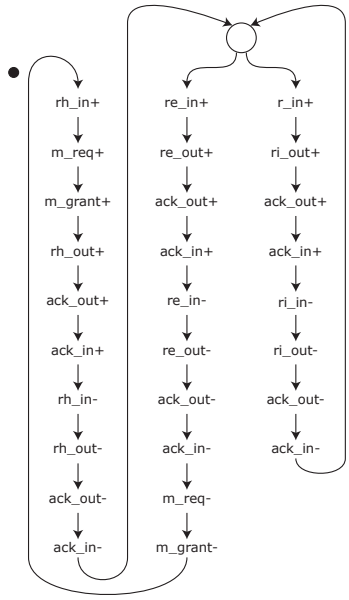


Figure 4.5: STG specification of the access control circuit.

header, intermediate, and end request signals are denoted *rh*, *ri*, and *re* respectively. The mutex request and grant signals are denoted *m_req* and *m_grant*. The fairness of the arbitration is determined by the mutex component.

Delay elements are inserted on the request signals on the output channel. The delay elements must match the delay that the data signals experience in the merge component subtracted by the delay through the access control circuit.

4.2.3 Merge

The merge component has four input channels and one output channels. It relays handshakes from the input channels to the output channel. It is assumed that input requests are mutually exclusive. The design of the merge component is shown in figure 4.6.

The design is based on the ordinary merge design presented in [24], but it has been modified to support the three-requests handshake channel. For each input channel the three request signals must be OR'ed together. The output of the OR gate and the output ack signal is used to generate the input ack signal

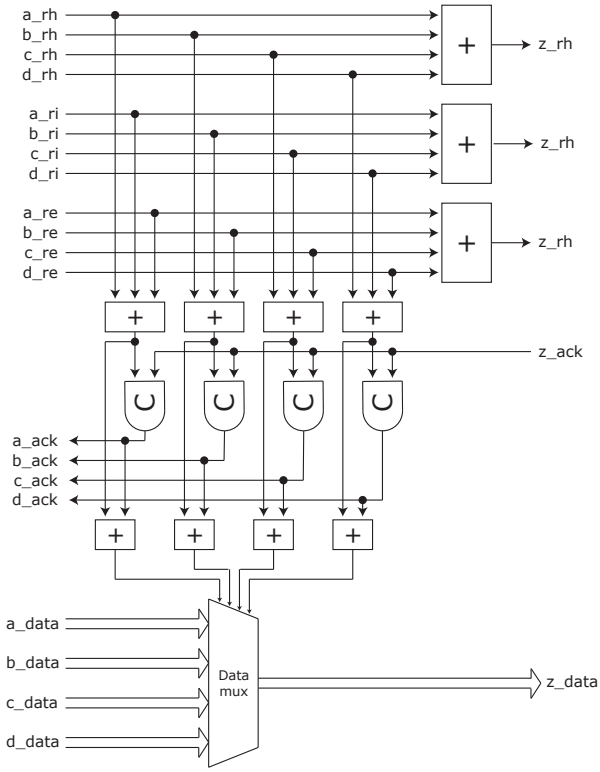


Figure 4.6: The merge component.

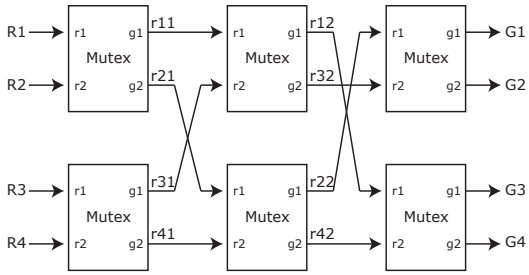


Figure 4.7: The 4-input mutex.

using a C-element. The added overhead for the 4-input merge component to support the additional request signals is four 3-input OR gates and two 4-input OR gates.

To support broad data validity the request signals are OR'ed with the acknowledge signals. This ensures that the data multiplexer selects the active input for the complete handshake cycle.

4.2.4 Mutex

A 4-input mutex component is needed for the output port design. A 4-input mutex can be constructed by combining several 2-input mutex components. QNoC [23] also utilizes a 4-input mutex component and their design is also used in this project. The 4-input mutex component consists of six 2-input mutex components arranged in three stages. The design is shown in figure 4.7. In [23] an analysis of the fairness of the design is carried out. They proof that the mutex has a bounded blocking time and a request may be outrun by no more than two later requests. The proof assumes that the 2-input mutex components are fair. Even though the mutex will not preserve the original ordering in all cases, it is considered to be fair enough for the purpose of this project, since assuring fairness is not a key issue.

4.2.5 Fifo Buffer

FIFO buffers are inserted at each input and output port. The FIFO is designed, in the regular way, as a chain of handshake latches. This is shown in figure 4.8(a).

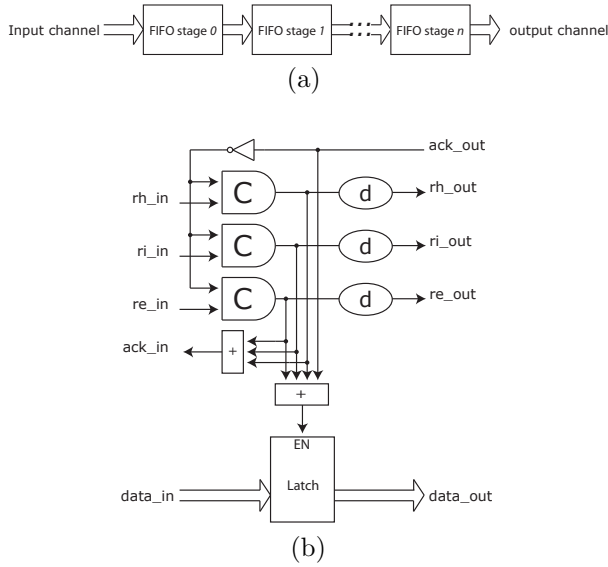


Figure 4.8: (a) A FIFO consists of a chain of FIFO stages. (b) the design of an un-decoupled FIFO stage.

Handshake latches for the 4-phase bundled data protocol can be designed in three different ways, depending on how strong the coupling is between the input channel and the output channel [24]: *Un-decoupled*, *Semi-decoupled*, and *Fully-decoupled*. The selection between the different types is a tradeoff between complexity and performance.

The un-decoupled latch controller is the least complex. It does not allow latching of new data before the previous handshake cycle has finished completely, i.e. it must wait for $Ack_{out}\downarrow$. In other words, it must wait for the superfluous return-to-zero phase of the handshake to finish. There exists a strict ordering between the handshaking on the input channel and the output channel: $Req_{out}\uparrow \preceq Ack_{in}\uparrow$ and $Req_{out}\downarrow \preceq Ack_{in}\downarrow$. During the return-to-zero phase the latch is transparent. Consequently only every second latch in a FIFO will hold valid data. It is said to have a *Static spread* of 2. Also, due to dependencies with non-neighboring stages in the FIFO, it is unable to take advantage of an asynchronous delay element.

The semi-decoupled latch controller allows every latch in a FIFO to hold valid data, by allowing new data to be latched after $Req_{out}\downarrow$. This is achieved by relaxing the ordering of the handshaking between the input channel and the output channel to $Ack_{out}\uparrow \preceq Ack_{in}\uparrow$. The Static spread is 1. Like the un-decoupled it is not able to take advantage of an asynchronous delay element.

The fully-decoupled latch controller has a Static spread of 1 and is able to take advantage of the asynchronous delay element. This is achieved by allowing new inputs to be latched after $Ack_{out}\uparrow$. Thus, the handshaking between the input channel and the output channel is completely decoupled.

Despite the performance advantage of the more advanced latch controllers, an un-decoupled latch controller is used in the design of the FIFO. The main reason for this is its simplicity and the fact that performance does not have high priority in the project.

The design of the handshake latch is shown in figure 4.8(b). The design is a muller pipeline handshake latch (figure 2.2(c) p. 7) extended with the extra request signals and broad data validity. The latch is a level sensitive latch that is transparent when enable is 0 and opaque when enable is one. The handshake latch accepts early data validity and produces broad. A C-element is used for each request signal, and the Ack_{in} signal is generated by OR'ing the outputs of the C-elements. The latch control signal is generating by OR'ing the outputs of the C-elements with the Ack_{out} signal to provide broad data validity. The OR'ing with Ack_{out} assures that the latch is kept opaque for the complete handshake phase.

4.3 Network Adapter Design

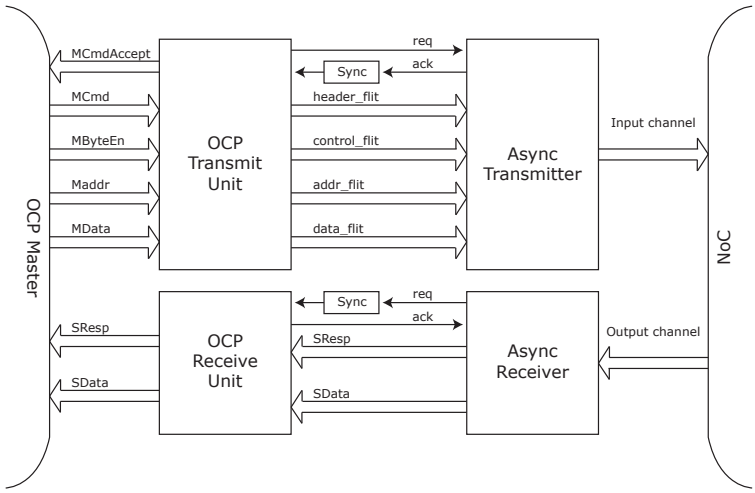
The NA handles the communication between the cores and the network. The NA consists of a Core Interface (CI) and a network interface (NI). The Core Interface (CI) is a memory-mapped interface and the Network Interface (NI) is a message-passing interface. As stated earlier The OCP protocol is used at the CI.

An OCP compliant NA for MANGO is presented in [5]. In [17] an improved NA for MANGO is presented. Due to the inclusion of GS in MANGO, the NAs are more complex than what is needed for this project. Therefore, a more simple design has been made for the NoC.

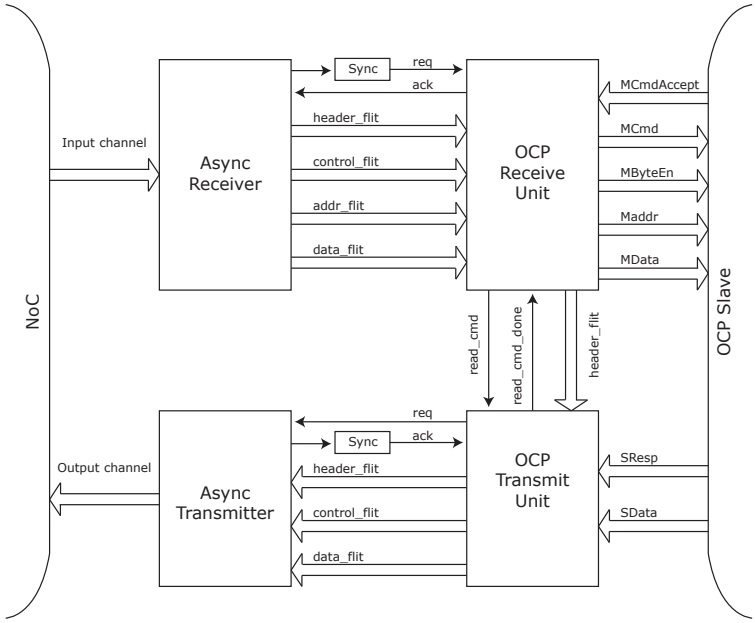
The NA design consists of a Master NA and a Slave NA. The Master NA is used with a OCP master core and the Slave NA is used with an OCP slave core. The NAs must translate an OCP command into a network packet. The three different packet types was shown in table 4.2 on page 55. The Write Request and Read Request packets are four flits long, while the Read Response packet is three flits long. To simplify the design the the NA, it will always transmit a four flit package. For the Read Response it will append an empty flit.

When designing the NA the placement of the crossing between the synchronous and asynchronous domain is important. The number of clock synchronization in a design should be kept as low as possible, for two reasons: speed and reliability. Each time a signal is passed through a synchronizer it takes two clock cycles (for a two-flip-flop synchronizer) and thereby decreases the speed. The possibility of metastability can never be removed completely, thus the possibility of failure will always exist. The failure rate will increase with the number of synchronizations performed. The only way to reduce the possibility of metastability is to increase the latency by adding more flip-flops in the synchronizer. Therefore the NA is designed so it is only necessary to perform synchronization one time per packet transmission.

The design of the master NA and the slave NA is shown in figure 4.9. The following sections will present the design of the master and slave NA.



(a) Master NA



(b) Slave NA

Figure 4.9: The design of the master and slave NA.

4.3.1 Master NA

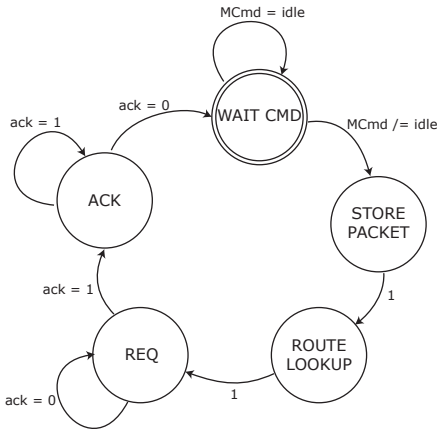
The master NA consists of a transmitter part and a receiver part. The transmitter part has a synchronous CI, the *OCP Transmit Unit*, and an asynchronous NI, the *Async Transmitter*. Likewise the receiver part contains the *OCP Receive Unit* as CI and the *Async Receiver* as NI. The synchronous and asynchronous circuit communicates using the 4-phase bundled data handshake protocol. The acknowledge that is sent from the asynchronous domain into the synchronous domain is passed through a synchronizer as the one presented in section 2.5.4 (p. 22).

The OCP Transmit Unit handles the communication with the OCP interface. The OCP Transmit Unit latches the OCP command and presents it in the four-flit package format on its output. The OCP Transmit Unit will determine the route to the destination by doing a lookup in a hardcoded ROM based on the 4 MSBs of the destination address (not shown in the figure). When the packet data is ready it asserts the request signal for the Async Transmitter. The Async Transmitter performs the serialization of the packet flits onto the network. When it has finished the transmission, it will finish the handshake with the OCP Transmit Unit and the master NA is ready for a new transaction.

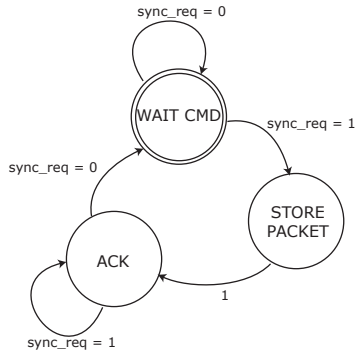
A more advanced design of the route lookup could be implemented using a Content Addressable Memory (CAM). This will reduce the memory required to implement the lookup table, especially for systems with many cores.

The receiver part listens on the NI interface waiting for a packet. When a packet arrives the Async Receiver latches the packet and asserts the request signal for the OCP Receive Unit. The OCP Receive Unit will present the packet at the OCP interface and afterwards finish the handshake with the Async Transmitter.

The OCP Transmit Unit and OCP Receive Unit is designed as mealy type state machines and the state diagrams is shown in figure 4.10. The STG specification of the Async Transmitter and Async Receiver is shown in 4.11. Note that the Async Receiver is able to receive both 3-flit packets and 4-flit packets, but only 4-flit packets are transmitted in the network. The Async Receiver must be able to handle that a new header request is received before it has finished the handshaking with the OCP Receive Unit.



(a) OCP Transmit Unit

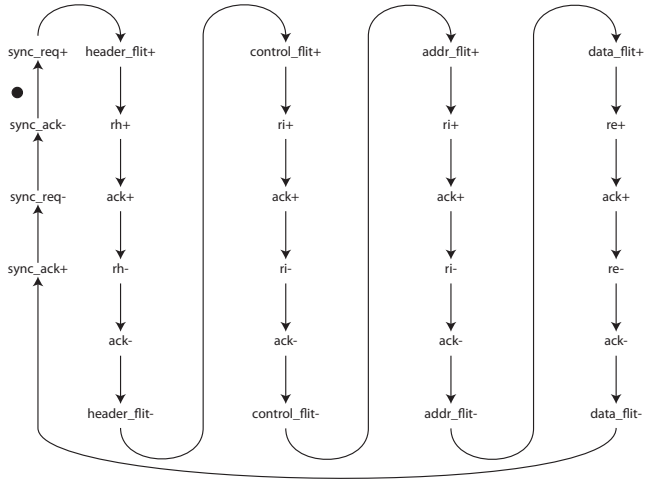


(b) OCP Receive Unit

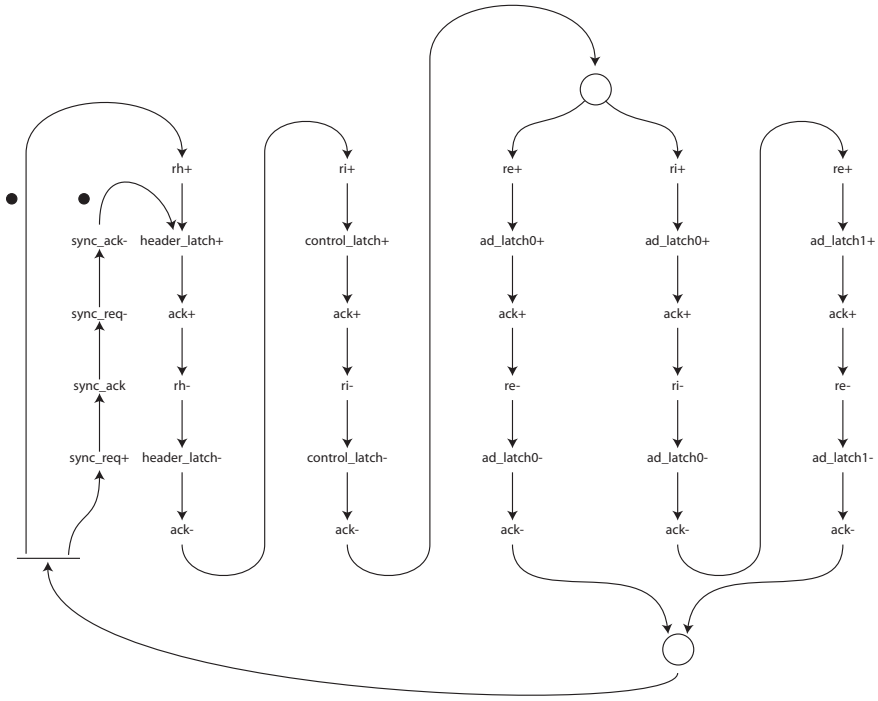
State	Action
Wait cmd	Wait for OCP Command.
Store packet	Set enable pin for packet register.
Route lookup	Wait for route lookup to finish, unset enable pin for packet register, and set SCmdAccept to finish OCP transaction.
Req	Set request to NI.
Ack	Unset request to NI.

State	Action
Wait cmd	Wait for new packet at NI. Unset acknowledge to NI.
Store packet	Set enable pin for packet register.
Ack	Set acknowledge to NI.

Figure 4.10: State diagrams for the Master NA.



(a) Asynch Transmitter



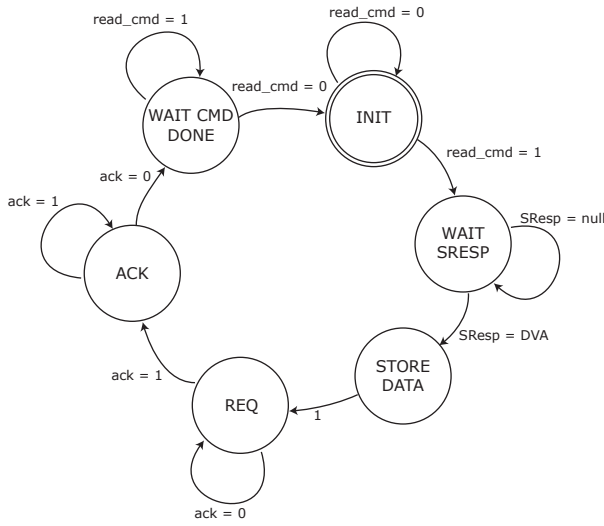
(b) Asynch Receiver

Figure 4.11: STG specifications for the asynchronous transmitter (a) and for for the asynchronous receiver (b).

4.3.2 Slave NA

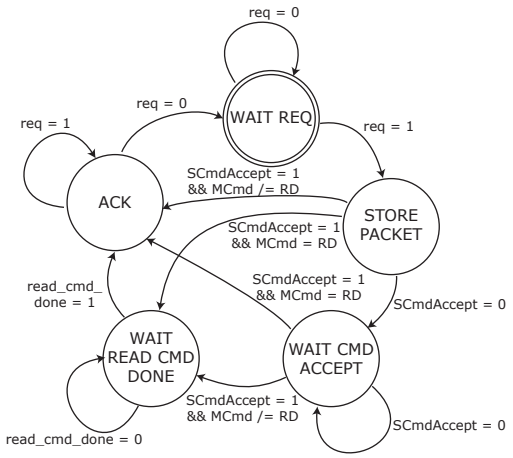
The design of the slave NA is very similar to the design of the master NA as can be seen from figure 4.9. However subtle differences exists. In case of a Read Request packet, the slave NA must not accept new packets at its NI receive interface before it has transmitted the Read Response packet onto the network. Therefore the OCP Transmit Unit must notify the OCP Receive Unit when it has completed the transmission. Furthermore the OCP specification requires that the receiver of a response is always ready since the response data is only valid for one clock cycle. To fulfill these requirements the OCP Receive Unit and the OCP Transmit unit will in case of a Read Request perform a handshake using the *read_cmd* and *read_cmd_done* signals. Also the return path data from the Read Request is sent to the OCP Transmit Unit.

The state diagrams for the OCP Transmit Unit and the OCP Receive Unit is shown in figure 4.12 and in figure 4.13 respectively. The Async Transmitter and Async Receiver circuits are identical with the circuits used in the master NA.



State	Action
Init	Wait for a read cmd from NA receiver. Unset read_cmd_done.
Wait SResp	Wait for OCP response cmd.
Store data	Set enable pin for packet register.
Req	Set request to NI and unset enable pin for packet register.
Ack	Unset request to NI.
Wait cmd done	Set read_cmd_done.

Figure 4.12: State diagrams for OCP Transmit Unit in the Slave NA.



State	Action
Wait req	Wait for new packet at NI. Unset register enable and acknowledge.
Store packet	Set register enable.
Wait cmd accept	Wait for OCP slave to accept OCP cmd.
Wait read cmd done	Set read_cmd and wait for read_cmd_done.
Ack	Set acknowledge to NI and unset read_cmd.

Figure 4.13: State diagrams for OCP Receive Unit in the Slave NA.

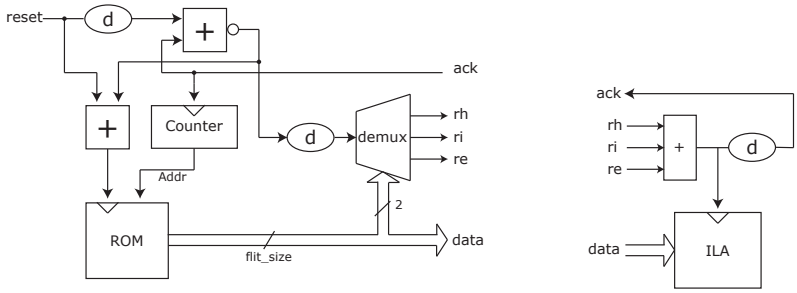
4.4 Traffic Generator Design

For testing purposes a simple traffic generator has been designed. The design consists of a traffic source and a traffic sink. The traffic source is able to transmit a predefined set of packets from a ROM. The traffic sink reads the received packets into a ROM. It is then possible to compare the input trail with the output trail to verify correct operation.

The design of the traffic source is shown in figure 4.14(a). Each entry in the ROM consists of a flit type and flit data. The handshaking is done using a simple repeater circuit (a Haste repeater [25]) that consists of a single NOR gate. A synchronous counter clocked on the acknowledge signal is used to increment the address input of the ROM. The ROM is clocked with the un-delayed request signal. For proper initialization of the ROM output the clock input is gated with the reset signal. The flit type is used to control a de-multiplexer to output the correct request type.

The traffic sink must read the received packets and store them in a ROM. When the complete input trail has been received the sink ROM data must be read out of the FPGA. This is possible through the JTAG interface, however Xilinx does not provide any simple tools that can do that directly. Manual communication with the JTAG interface is required to extract the data.

Fortunately, the ChipScope tool provided by Xilinx [33] can be used to read the data into the ROM and extract it afterwards. In other words, it can almost build the complete sink component. ChipScope is a complex logic analyzer tool for Xilinx FPGAs. It provides cores that can monitor and store data traces of any signal in the FPGA during runtime. The ChipScope software is used to



(a) Traffic Source (b) Traffic Sink

Figure 4.14: Traffic generator design

extract and display the data captured by the cores. The ChipScope software has a GUI interface and is relatively easy to operate.

For the sink design the ILA (Integrated Logic Analyzer) ChipScope core is used to capture data. The core captures the data on the data signal on the falling edge of the request signal. When the internal storage of the ILA core is filled, the data is transmitted to the ChipScope software. The design is shown in figure 4.14(b). An ILA core is needed for each signal that must be monitored in the design.

Chapter 5

Asynchronous Network-on-Chip Implementation

This chapter describes the implementation of the designed NoC. The implementation of the NoC components follows the design flow presented in chapter 2. Therefore this chapter will only give few relevant comments to the implementation of the different components.

In section 5.1 the implementation of the router is described and in section 5.2 the implementation of the NAs are described. Finally the traffic generator implementation is described in section 5.3.

5.1 Router

The implementation of the router is divided into 9 VHDL entities which follows the structure of the design presented in section 4.2. The VHDL files implementing the router is found in appendix A.5.2. Below each VHDL entity is listed along with a short description.

`be_router`:	The top-level router entity. Connects all the ports and FIFOs with each other. The depth of the input and output FIFOs are set to 1. Appendix A.5.2.1 p. 132.
`fifo`:	The FIFO. The depth is configurable using a VHDL *generic*. Appendix A.5.2.2 p. 145.
`fifo_stage`:	Implements the FIFO stage. Appendix A.5.2.3 p. 147.
`input_port`:	Implements the Input Port. Appendix A.5.2.4 p. 149.
`header_rotater`:	Subcomponent of the input_port. Implements the data multiplexer and the control circuit. Appendix A.5.2.5 p. 152.
`output_port`:	Implements the Output Port. Appendix A.5.2.6 p. 153.
`access_control`:	Implements the STG for the access control circuit. The Petrify equations is included in the source file. Appendix A.5.2.7 p. 158.
`mutex4`:	Implements the four-input mutex component. Appendix A.5.2.8 p. 160.
`merge4`:	Implements the merge component. Appendix A.5.2.9 p. 162.

The router is implemented with a configurable flit size. The flit size is defined in the `types.vhd` file along with other global constants. `types.vhd` is found in appendix A.5.5.8 (p. 254).

The encoding of the routing direction used in the header flit is the following:

```
North = "00"
 East = "01"
South = "10"
 West = "11"
```

As mentioned in the design, the local port is reached by routing the flit back in the same direction it came from.

The area utilization of each component is listed in tabel 5.1. The number of utilized LUTs is excluding delay elements. Thus the total area utilization of the router is 1295 LUTs and 330 latches The percentage of the LUTs that are used for delay matching is 29%.

Component	LUTs	Latches	Delay elements
FIFO stage	3	32	15
Input port	47	2	22
Output port	130	0	24
Access Control	5	0	0
Mutex	24	0	0
Merge	86	0	0
Router	915	330	380

Table 5.1: Area utilization of the router components.

5.2 Network Adaptor

The implementation of the Master NA and Slave NA is divided into 10 VHDL entities. The implementation follows the structure from the design presented in section 4.3 on page 64. The VHDL files implementing the components are found appendix in A.5.3. The entities is listed below along with a short description:

Master NA

`master_na`:	The top-level entity. Appendix A.5.3.1 p. 172.
`ocp_master_transfer_unit`:	The transmit CI. Appendix A.5.3.2 p. 175.
`ocp_master_receive_unit`:	The receive CI. Appendix A.5.3.3 p. 178.
`route_lookup_tables`:	VHDL package with constants that specifies the route lookup tables. Appendix A.5.5.9 p. 255.

Slave NA

`slave_na`:	The top-level entity. Appendix A.5.3.4 p. 179.
`ocp_slave_transfer_unit`:	The transmit CI. Appendix A.5.3.5 p. 183.
`ocp_slave_receive_unit`:	The receive CI. Appendix A.5.3.6 p. 185.

Common for both NAs

`async_transmitter`:	The transmit NI. Appendix A.5.3.7 p. 188.
`async_transmitter_hs_ctrl`:	Implements the STG for the control circuit for the transmit NI. The Petrify equations is included in the source file. Appendix A.5.3.8 p. 190.
`async_receiver`:	The receive NI. Appendix A.5.3.9 p. 197.
`async_receiver_hs_ctrl`:	Implements the STG for the control circuit for the receive NI. The Petrify equations is included in the source file. Appendix A.5.3.10 p. 199.

The route lookup table in the Master NA is implemented as a ROM using the Block RAM resources available on the FPGA. Block RAM can be included in two ways: inferred by HDL or instantiated as an IP core generated by the Xilinx Core Generator. When the ROM is inferred by HDL it is much easier to change the content of the ROM. Therefore that approach has been chosen. The initialization values for the ROM is included in the file `route_lookup_tables.vhd` as

Component	LUTs	Latches	Delay elements
Master NA	197	116	12
Slave NA	115	220	12

Table 5.2: Area utilization of the Master NA and Slave NA.

VHDL constants. The initialization values are passed to the Master NA entity as a VHDL generic.

In the implementation of the `async_transmitter` STG the C-elements `csc1` and `csc2` must be initialized to 1 during reset to avoid a glitch on the acknowledge signal for the OCP Transmit Units (`sync_ack_in`). The reset value of the two C-elements is not included in the list of set/reset values in the implementation specification by Petrify. In the reset state of the circuit the inputs to the two C-elements should set their outputs to 1. If they are reset to 0 the glitch is introduced in the moment the reset is removed.

The `async_transmitter` receives a *packet_type* signal. This is not used since it always transmits four flits.

During post place and route simulations a glitch is observed on the sync_ack signal generated by the Receive CI for both NAs. The signal is used in asynchronous components, so it must be hazard free. The glitch happens because the signal is not directly connected to the output of a flip-flop, but passes through combinatorial logic. The glitch is removed by the insertion of a de-glitch flip-flop.

The area utilization of the Master NA and the Slave NA is listed in table 5.2. The number of utilized LUTs is excluding delay elements. Also, Block RAM usage is not included in the table.

5.3 Traffic Generator

The implementation of the Traffic Generator is divided into 2 VHDL entities. The implementation follows the structure from the design presented in section 4.4. The VHDL files implementing the components are found in appendix A.5.4. The entities are listed below:

`traffic_source`:	The Traffic Source. Appendix A.5.4.1 p. 204.
`traffic_sink`:	The Traffic Sink. Appendix A.5.4.2 p. 206.
`source_rom_data`:	VHDL package that specifies the flit data used by the Traffic Source. Appendix A.5.4.3 p. 207.

The flit data is implemented in a Block RAM based ROM using the same approach as in the NA and the flit data is specified in the file `source_rom_data.vhd`.

The ChipScope ILA core that is used in the sink to collect the data can be inserted into the design in two ways. The ChipScope software can generate the ILA cores which can be instantiated in the design in the normal way. ChipScope can also insert the cores in the design by inserting them in the synthesized net-list automatically. The last method is a lot easier since no changes is needed in the VHDL code. However, the last method can not be used in this case. The synthesizer will remove the data signals from the handshake channel because they are not connected to anything in the traffic sink, thus they will not be in the synthesized net-list. Therefore the ILA cores are instantiated manually in VHDL. Along with the ILA core an ICON control core must also be inserted in the design. The ICON core handles the communication with the ChipScope software over the JTAG interface.

The falling edge of the request signal is used as clock input for the ILA core. This signal must be routed on the dedicated clock nets for the core to work. This is done by inserting a clock buffer (BUFG) on the signal.

The Traffic Sink contains a delay element to delay the acknowledge signal. If this delay is not sufficiently large the ILA core will not work properly. A size of 10 is found to work. The ChipScope documentation says it supports frequencies of up to 500 MHz, thus a delay should not be required. It is expected that the ILA core fails because it expects a "real" clock signal but it is clocked with the request signal that does not have a regular period.

The ChipScope cores contains a bug so that they will not work with bus width larger than 16. The error only happens with some designs and is not officially recognized by Xilinx. The bug results in an DRC error during the mapping process. In another post in the Xilinx Community Forums the same issue is reported [36]. It has not been possible to determine the exact source of the error.

Chapter 6

Asynchronous Network-on-Chip Test

6.1 Introduction

The NoC components are tested to ensure that they work as intended. The individual components are tested by post place and route simulations in Modelsim. The router is also tested by running an on-board test using ChipScope. The primary goal of the tests is to documents that the components works, but performance is also briefly evaluated.

For component simulation a source and sink simulation component is used to generate traffic. The VHDL components are found in appendix A.5.6.1 and A.5.6.2. For the simulation and on-board test of the router the traffic generator presented in section 4.4 is used.

6.2 FIFO

The FIFO is simulated by attaching a source to the input and a sink to the output. The depth of the FIFO is set to 12, so a total of 6 valid tokens can

be in the FIFO simultaneously. The FIFO is the only state-holding component in the router, so the throughput of the FIFO sets an upper bound for the throughput of the router. To measure the maximum throughput the source/sink produces/consumes tokens as fast as possible, i.e. there is 1 ps between the receival of the acknowledge/request, to the assertion of the request/acknowledge.

The period, P, of the FIFO is the delay between the input of a valid token and the input of the next valid token [24]. The period is found by measuring the delay between subsequent assertions of the request header signal and then divide by three. This is done because there are small variations in the period for the three request types, thus the average period is measured.

The period of the FIFO is found to

$$P_{FIFO} = 37.2/3 = 12.4ns$$

Which is equivalent to a throughput of 80.6 MHz.

The Modelsim print of a part of the simulation is found in appendix A.2.1

6.3 Input Port

The purpose of the simulation is to validate that the Input Port latches the routing direction from the header flit correctly and that the header is correctly rotated. A source is attached to the input and a sink is attached to each output. The source sends a 3-flit packet targeted for each output.

The Modelsim print of the simulation is found in appendix A.2.2.

6.4 Output Port

The purpose of the simulation of the Output Port is to validate that the arbitration between contending inputs works correctly. A source is attached to each input and a sink is attached to the output. The sources asserts their header request signal simultaneously and the Output Port should arbitrate between the inputs and let them through to the output one at a time. Each source sends a 3-flit packet. The Perl script to correct the mutexes must be used to run the simulation.

The Modelsim print of the simulation is found in appendix A.2.3.

6.5 Router

The router is tested with the Traffic Source and Traffic Sink presented in section 4.4. This design is tested both in simulation and on the FPGA using ChipScope to collect the data at the sinks. These tests are done with a flit size of 16 bits, due to the problems with ChipScope and large busses mentioned in section 4.4. The ILA cores must be instantiated in the top-level VHDL component. The router entity with inserted ChipScope cores can be found in appendix A.5.2.10 (p. 164).

A source is connected to each input FIFO and a sink is connected to each output FIFO of the router. The depth of the FIFOs is 1. The sources repeatedly transmits a 3-flit packet to the Output Ports in the following sequence:

$$North \rightarrow East \rightarrow South \rightarrow West \rightarrow North \rightarrow \ldots$$

When the destination is the same as the source, the packet will be routed to the local port. In this way it is tested that the router routes the packets in the correct direction and also arbitration is tested. The source data for the traffic generator is found in appendix A.5.4.3 (p. 207).

The Modelsim print of the simulation is found in appendix A.2.4 (p. 112) and the data captured by the ChipScope sinks is found in appendix A.2.5 (p. 116). The ChipScope plot is exported to the VCD file format and displayed using Modelsim. Only the beginning of the test is included in the plots.

For the simulation it is not needed to use the Perl script to correct the mutexes. Due to different wire delays the requests arrives with a large enough temporal distance so that the mutexes does not start to oscillate.

The ChipScope test does not give any temporal information of how flits traverses the net. It only lists the sequence with which the flits arrives at the sinks.

It is interesting to note that the sequence with which packets arrives at the sink is different for the simulation and the on-board test. This must be due to different arbitration result.

The period with which the sinks consumes flits is approximately

$$P_{router} = 70/3 = 23.3ns$$

This is equivalent to a throughput of 43 MHz. The period varies with about 1-2 ns between the output ports.

6.6 Network Adaptor

The NAs are tested in simulation by connecting a Master NA and a Slave NA through the NIs. A simple OCP master simulation module is connected to the OCP interface of the Master NA and a simple OCP slave simulation module is connected to the OCP interface of the Slave NA. The simulation modules is found in appendix A.5.6.3 and A.5.6.4. The OCP master issues a write command followed by a read command. The OCP slave accepts the write command and issues a response to the read command which is sent to the OCP master.

Note that this test is rather simplified, e.g. it does not test the case where another request arrives while processing a previous request.

The Modelsim print of the simulation is found in appendix A.2.6.

Chapter 7

Asynchronous NoC-Based MPSoC Prototype

7.1 Introduction

A small GALS type multi-processor SoC prototype has been developed to demonstrate the NoC. The system is based on a previous system developed in the project *A NoC-based SoC Executing a Ray Tracer, using Synchronous Multiprocessing* at IMM, DTU [22]. In that project an FPGA implementation of a synchronous NoC-Based SoC that executes a distributed ray tracer application is presented. The primary goal with the prototype is to demonstrate the NoC, hence the application which is executed on the system is less important. By choosing an existing system as the basis for the prototype the "plug'n'play" functionality of the NoC is demonstrated.

7.2 Synchronous NoC-Based SoC

This section will present the cores used in the synchronous NoC-Based SoC presented in [22], which is used as the basis for the MPSoC prototype.

The SoC consists of multiple CPUs communicating using a shared memory space, in which peripherals are mapped into. The system contains five CPUs, two RAMs, a semaphore unit, and a UART. OCP is used as the interface between the cores and the NoC.

The OpenRISC 1200 (OR1200) CPU from OpenCores [19] is used in the system. It is a 32-bit processor with a 5-stage pipeline. The CPU has an optional cache system, that has been disabled.

Synchronization methods are not supported by the OR1200, so a memory mapped semaphore unit provides the required synchronization methods. The semaphore is acquired using OCP read requests, and write requests are used to release the semaphore. The result of a semaphore request is transmitted in a read response, i.e. has the semaphore been acquired or not. A busy waiting scheme is used, thus a core must keep polling the semaphore unit until the semaphore is acquired. This generates a lot of unnecessary traffic, but is very simple to implement.

The UART is used to communicate with the outside world over a serial interface. The UART implementation is also from OpenCores.

The RAMs are implemented using the Block RAM resources available on the FPGA.

The UART and OR1200 is from OpenCores. Therefore they use the WISHBONE interface. OCP wrappers for the cores are used to convert from the WISHBONE interface. The OR1200 has separate data and instruction memory interfaces. The OCP wrapper merges these into one interface, so it can be connected to the NoC using a single socket. An OCP wrapper is also provided for the Block RAM interface.

Two applications has been developed for the system. The first is a simple "Hello World" program where each CPU sits in an infinite loop and writes messages to the UART. No data is exchanged between the CPUs. The semaphore for the UART is acquired and released for each message. This program can be run with 1-5 CPUs and only 1 RAM. The second application is the full ray tracer application which requires 5 CPUs and 2 RAMs.

The system is implemented on a Xilinx Virtex-II FPGA.

7.3 MPSoC Overview

As much as possible from the base system have been reused in the implementation of the prototype. Because both systems are OCP based, all the cores can be reused with no modifications. Also, the source code for the applications can be reused without modifications. Naturally the type of interconnect is invisible from the software abstraction level.

Due to exhaustion of resources on the FPGA it has not been possible to fit the complete system on the FPGA. It has only been possible to fit a system with three CPUs, one RAM, semaphore, and UART on the FPGA. The possible utilization percentage of the FPGA resources is lower than expected. This is more thoroughly explained in section 7.5. As a consequence only the "Hello World" application can be executed on the system. This is a very simple application, so parallelism in the NoC is not so thoroughly demonstrated as it would have been if the complete system could be used. However, it is still sufficient to demonstrate the NoC.

7.4 MPSoC Design

For the design of the MPSoC system the topology is designed to be used with the complete system. Therefore the topology is designed for a system with five master cores and 4 slave cores. It is required that all master cores are able to communicate with each of the slave cores. The cores are connected in a 3x3 mesh topology as shown in figure 7.1.

The UART and the semaphore is both connected to router 5. The UART is connected to the *local* port, while the semaphore is connected to the unused *east* port. The semaphore is not connected to the unoccupied router 6 because it makes it simpler to ensure deadlock freedom in the system.

Deadlocks may occur in the system due to dependencies in the network and due to dependencies between the cores in the system. In [12] deadlock problems in NoC-based system is covered.

Network deadlocks occurs due to cyclic dependencies in the network. As explained in section 3.2.3, they can be solved by different methods, e.g. by using *xy*-routing. Even if a system is free from network deadlocks, the complete system may not be deadlock free. Deadlock freedom in the network assumes that when a packet reaches the destination-NI, the NI will eventually consume the

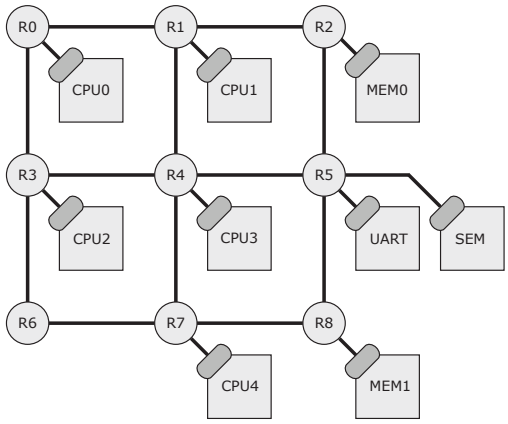


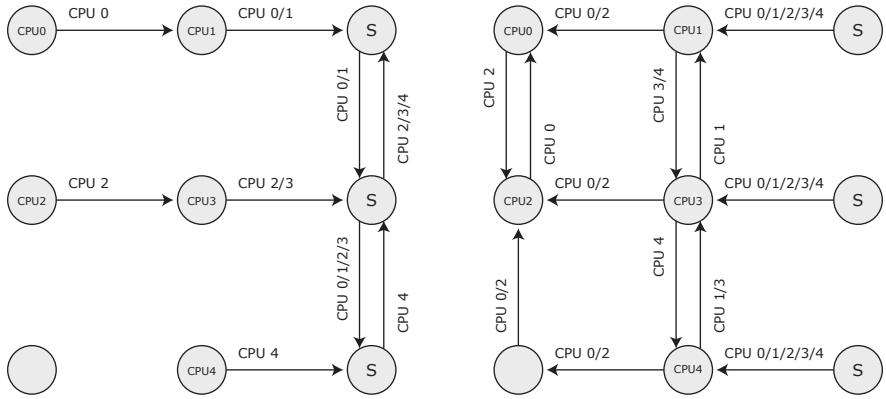
Figure 7.1: Topology with five CPUs, two RAMs, one UART, and one semaphore.

packet. If this is not the case, the system may deadlock. This situation can happen due to message-dependencies between the cores in the system.

In this system request-response message-dependencies may exist between the master cores and the slave cores. A request-response dependency arises if an incoming request at the slave NI blocks the transmission of a response. The buffers behind the blocked request, will fill up and the system will deadlock. Request-response dependencies can be solved by ensuring that enough buffer space is available at the NIs. The buffers should be so large, that they will never fill up. The CPUs used in this system has maximally one outstanding request, because Read requests are blocking. If all CPUs issues a write request followed by a read request destined for the same core, a maximum of 10 packets is possible at each Slave NA. Thus, the buffers at the Slave NAs must fit at least 10 packets, to assure that they never will fill up.

The dependencies can also be resolved by using independent request and response networks. Separating the request and response networks is done by only allowing each (unidirectional) link to transfer *either* request packets *or* response packets. Using this approach an incoming request is never able to block the transmission of a response. The separation of channels can be done either physically or logically by using virtual channels. The individual request and response networks must still be free from routing deadlocks.

To ensure deadlock freedom in the MPSoC system, physically separated request and response networks are used. If xy-routing is used in the independent request



(a) Request net (b) Response net

Figure 7.2: Request and response net.

Peripheral	Address Space
UART	0x80000000 - 0x8fffffff
Semaphore	0x40000000 - 0x4fffffff
Memory	all others.

Table 7.1: Assigned address spaces for peripherals.

and response networks, the system is ensured to be deadlock free, as explained above. In figure 7.2 the separated request and response networks for the system is shown. Each link is marked with the master cores that uses it.

It is sufficient that the buffer capacity of each router is one flit. Therefore the depth of the input and output FIFOs is set to 1. Due to the un-decoupled latch controllers two handshake latches are needed to store one flit.

For the "Hello World" application the *MEM1* memory core in figure 7.1 is not needed. Thus, the system has three memory mapped peripherals. The assigned address space for each peripherals is listed in table 7.1.

7.5 MPSoC Implementation

A design with three CPUs, one RAM, one UART, and one semaphore running the "Hello World" application is implemented. To be able to fit the design on the FPGA the mesh is reduced to a 3x2 mesh. From figure 7.1 the implemented

system contains the following cores CPU0, CPU1, CPU2, MEM0, UART, and SEM. The routers R6, R7, and R8 has been removed. With only three CPUs and one RAM in the system the three removed routers is not used anyway, so the original routing within the topology can be kept.

The "Hello World" application is implemented in C and the source code is found in appendix A.6.

The following VHDL entities are used in the implementation of the prototype:

`MPSoC_noc`:	The top-level entity. Appendix A.5.5.1 p. 209.
`noc_mesh`:	A 3x2 mesh of routers. Appendix A.5.5.2 p. 225.
`or1200_ocp`:	OCP wrapper for the OR1200. From [22]. Appendix A.5.5.3 p. 239.
`or1200_mem_if`:	Used in the OR1200 OCP wrapper to merge the instruction and data interface. From [22]. Appendix A.5.5.4 p. 244.
`core_mem_ocp`:	OCP interface for Block RAM cores. From [22]. Appendix A.5.5.5 p. 247.
`semaphore_ocp`:	Semaphore unit. From [22]. Appendix A.5.5.6 p. 249.
`uart16550_ocp`:	OCP wrapper for the UART core. From [22]. Appendix A.5.5.7 p. 250.
`types`:	VHDL package that specifies global constants. Appendix A.5.5.8 p. 254.
`MPSoC_noc.ucf`	The User Constraints File (UCF) where the clock and `tig` constraints are assigned. Appendix A.5.5.10 p. 255.

Not included in the list above is the entities for the OR1200 and UART core from OpenCores and the entities implementing the routers and NAs presented in chapter 4.

To make a GALS-like system, two different clocks are used: $clk_1 = 40$ MHz and $clk_2 = 16.6$ MHz. The FPGA development board has a single 100 MHz oscillator. It is equipped with an external clock divider that feeds the FPGA with three different clock frequencies. Because the clocks are derived from the same oscillator, they cannot be considered to be completely independent. The clocks are further divided using the Digital Clock Managers (DCMs) available on the FPGA. clk_1 is used for CPU0 and CPU1 and clk_2 is used for CPU2 and the slave cores.

CPUs	Routers	LUT Util.	Run-Time
1	1	19%	12 mins.
2	1	30%	22 mins.
1	9	41%	23 mins.
3	6	65%	96 mins.
2	9	67%	270 mins.
3	9	73%	2430 mins.[a]

[a]Map failed. Utilization numbers are post-synthesis.

Table 7.2: Mapping run-time observations for different configurations, ordered by run-time.

To exclude the asynchronous components from timing analysis the `tig` constraint is used as explained in section 2.7.3. The `tig` constraint is applied to all signals in the `noc_mesh` entity using a wildcard in the UCF file. `tig` is also applied to the signals in the NAs that crosses from the synchronous to the asynchronous domain.

The reported maximum frequencies for the two clocks are:

$$clk_1 = 59.6 \text{ MHz}$$
$$clk_2 = 49.6 \text{ MHz}$$

The implementation of this down-scaled system utilizes 52% of the LUT resources.

As previously mentioned it has not been possible to fit the desired system on the FPGA. With increasing design sizes the run-time of the mapping process increases significantly and eventually it fails completely. A number of observations has been collected about the relation between the LUT utilization ratio and the mapping run-time for different configurations. These are shown in table 7.2. It has not been possible to fit a design that utilizes more than 67% of the LUTs. After the observations in table 7.2 was collected, an error was discovered in the implementation of the FIFO stage.[1] Correcting this error resulted in a significant decrease in the required delay sizes. With the reduced delay elements a design with 3 CPUs and a 3x3 mesh only uses 62% of the LUTs. This design is able to pass the mapping process but the place and routing process fails. Place and route fails with an error that the design is too dense.

It has not been possible to find any documentation from Xilinx that suggests whether these observations are normal or not. However, in a white paper pub-

[1]The error was previously mentioned in section 2.6.1 on page 25.

lished by Altera [3] (Altera is a competing FPGA supplier) they claim that they are not able to fit (synchronous) designs larger than 65% on a Xilinx Virtex-5 FPGA.

The observations by Altera suggests that there is a general problem with obtaining high resource utilizations on the FPGA. However it also seems that the NoC interconnect utilizes a high amount of the routing resources which lowers the possible utilization ratio even further.

7.6 MPSoC Test

The MPSoC prototype is tested by simulation and by an on-board test.

For the simulation three Modelsim prints are included:

1. CPU0 sends a Read Request and receives a Read Response.
2. The three CPUs are issuing Read Requests and are blocked by congestion on the network.
3. MEM0 receives a Read Request and sends a Read Response.

The shown parts of the simulation does not provide complete documentation of correct behavior of the system. It is only meant to illustrate key points in the simulation. It has unfortunately not been possible to make the Modelsim prints so it is possible to see the flit data.

In appendix A.3.1 the simulation of case 1 is showed. The CPU sends out a Read Request on the OCP interface. The Read Request is accepted by the Master NA and transmitted onto the network. The Master NA is blocked during the transmission of the end flit due to congestion on the network. After a while the Master NA receives the response and presents it on the OCP interface.

In appendix A.3.2 the simulation of case 2 is shown. The transmit NIs of the three CPUs are shown. They all sends Read Requests onto the network. During the transmission they are blocked by congestion on the network.

In appendix A.3.3 the simulation of case 3 is shown. The CI and NI of the MEM0 Slave NA is shown. The Slave NA receives a Read Request and presents the Read Request at the OCP interface to the MEM0 core. The core makes a respond and the NA sends a Read Response onto the network. From the simulation it can be seen that the Slave NA has a bug that causes it to present the same request again on the OCP interface after the transmission of the response. The core makes a response to the request but it is not transmitted onto the network. This bug does not cause any failures in this system but it might do in other systems.

For the onboard test the FPGA is connected to a PC using the serial interface. The port setting for the serial interface is set to 115200 bps, 8 data bits, one stop bit, and no parity bit. After programming and reset of the FPGA the messages from the CPUs is received on the serial interface of the PC. Figure 7.3

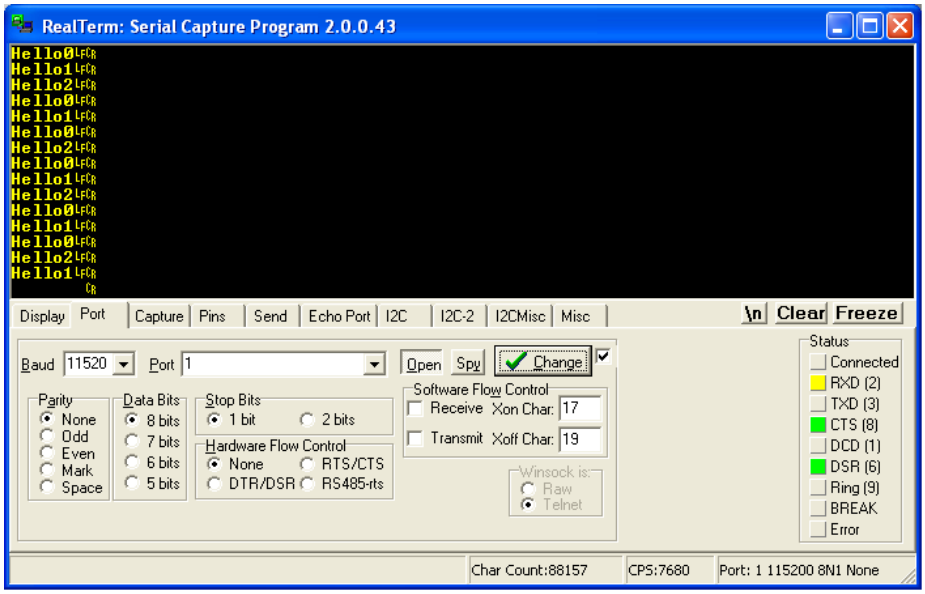


Figure 7.3: Screenshot showing the output when the "Hello World" program is running.

shows a screenshot from the terminal program when the "Hello World" program is running on the FPGA.

Chapter 8

Discussion

In this chapter the outcomes of the project will be evaluated along with proposals for possible future work.

8.1 Evaluation

The primary outcomes of the project is a general design flow for implementing asynchronous circuits on Xilinx FPGAs and an FPGA implementation of an asynchronous NoC. In the following these will be discussed.

The primary issues with implementing asynchronous circuits on FPGAs is to control the timing of the design. The delay elements have fluctuations in the produced delay and fluctuations have also been observed in the delay of the datapath. Acceptable predictability of the delay elements have been achieved by using placement constraints. It has proved to be harder to control the delay of the datapath. The methods that is used on synchronous designs to provide fine grained timing control of the datapath has not been found to be useful for asynchronous designs. In the project sufficiently large delays have been used to allow fluctuations in the delay of the datapath. In the implementation of the components the delay sizes are defined per component and is thus the same for all instantiations of the component. If tighter delay fitting is required, the

delays must be defined individually for each instantiation of the component. The delays can then be fitted in an iterative process. This will be very cumbersome to do without any tool aid. The Xilinx tools provides a method for performing incremental changes in the floorplan called SmartGuide. SmartGuide uses a previously place and routed design as a guide for the place and route process to achieve a similar placement and routing of the design. This method is not meant to be used after re-synthesis, but the Xilinx documentation says that it can be used if the synthesized netlists only differs slightly. For improved delay matching another design of the delay element can also be considered. A delay element where the size of the delay can be changed without altering the floorplan will minimize the effects of delay fluctuations in the datapath. A design with a locally clocked counter with a reset value stored locally using distributed RAM (LUTs used as RAM resources) will have this property. The counter could be clocked by an inverter chain implemented in LUTs. The design of such a delay element will most likely be larger than the relatively small delay chains used in this project but it will improve the performance.

In the implementation of the NoC latches are used to store data which is the normal approach used in asynchronous design because latches are less resource extensive compared to flip-flips. On an FPGA flip-flops and latches are equally expensive. By using flip-flops instead of latches it will be possible to relax some of the timing restrictions. For example, in the implementation of the input port the requirement of broad data validity can be relaxed by using a rising-edge triggered flip-flop to store the routing direction. In the FIFO stage flip-flips clocked on the rising edge of the request signal could also be used. This will provide broad data validity at the output of the FIFO stage without having to OR with the acknowledge signal. However this may require larger delay elements because the data will not be able to propagate through the flip-flop prior to the rising edge of the request signal, as it will in the case of a latch, due to the transparency of the latch.

In the design of the router a broad data validity scheme has been consequently used for all components, although it is only the input port that requires it. The FIFOs will accept early data validity so the input port and the output port does not have to provide broad data validity at their outputs for correct functionality.

To encode the flit type the NoC uses additional request signals in the handshake channel instead of appending the flit type to the flit data. This design simplifies the design of the router and also lower latency is achieved. However it also increases the complexity of the merge and FIFO stage components of the router because they must handle two extra request signals. Another disadvantage is that the number of delay elements increases from one to three for a handshake channel. Considering the issues with delay matching it may prove to be a better solution to append the flit type to the flit data. There has not been experimented

with the other approach so whether it is a better solution is unknown.

In the simulation of the prototype a bug was found in the Slave NA when Read Requests are processed. The bug do not cause any errors in the prototype, but if it is used with other cores it might lead to errors. In general it is believed that the design of the NAs can be improved in terms of both performance and area.

For the implementation of Petrify circuits two alternatives exists. Either a LUT-implemented C-element or a SR latch can be used as state-holding elements. In the project it was chosen to use C-elements for Petrify circuits and no issues have been encountered with this approach. On the other hand it should be noted that using SR latches theoretically is a more robust solution since it is a well-defined FPGA primitive in contrary to the LUT-implemented C-element.

During the implementation of the prototype it turned out that it was hard to reach a high utilization of the logic resources on the FPGA. It is unclear whether it is a general problem or specific for asynchronous design. There seems to be a general problem with obtaining high utilization ratios for the used FPGA but there are also indications of that the asynchronous implementation of the NoC utilizes a considerable amount of routing resources and thereby increases the utilization issues. Reducing the utilization of routing resources in the NoC can be done in mainly two ways: simplify the topology or reducing the flit size. The mesh topology used in the prototype is one of the simpler topologies. Alternatively a unidirectional torus could be used which will use about half of the resources compared to a mesh topology. The fewer links in the topology will result in higher utilization ratios of the links and thereby degrading performance. Also, it will be harder to ensure deadlock freedom in the topology without using virtual channels. By reducing the flit size the width of the handshake channels decreases, and thereby lowering the utilized routing resources. Also, the buffer sizes in the routers will be reduced. Naturally by decreasing the flit size the number of flits in a packet will increase thus, the packet latency through the network will increase. Another possibility of reducing the complexity of the system is to use simple traffic generators as traffic sources instead of full-fledged CPUs. To get a traffic generator to act like a CPU a more complex design than the traffic generators presented in this report is needed. The traffic generators should implement a state machine such that they can react dynamically to the traffic in the network.

The issues with low logic utilization affects the usability of the FPGA as a platform for prototyping asynchronous NoCs. Considering that it is a fairly simple NoC that has been implemented the possibility of prototyping more complex NoCs on the FPGA is degraded unless the utilization issues are solved. Prototyping more complex NoCs will require a larger FPGA. The version of the

Virtex-5 FPGA that have been used for the project is one of the smaller FPGAs in the Virtex-5 product line. The largest Virtex-5 has about 6 times as many slices. It can also be considered to try FPGAs from other manufactures.

8.2 Future Work

The implementation of the NoC is very basic, thus there are many possibilities for extending it. Some of the more interesting are listed below:

Virtual Channels An obvious extension to the NoC is the addition of VCs. All the design primitives to add VCs is available, so it is only a matter of adding additional buffers at the links along with some control circuitry. This addition should be a fairly trivial task.

Differentiated services A more complex extension is to extend the NoC to provide differentiated services. Differentiated services can be provided by having different service levels where high priority streams can take over lower priority streams. This can be implemented using VCs. To implement hard service guarantees connection-oriented routing must be used. In the MANGO NoC connection-oriented routing to provide hard guaranteed services are implemented by creating a virtual circuit using a series of connected virtual channels.

Fully de-coupled latch controllers The latch controllers used in the implementation of the FIFOs are simple un-decoupled latch controllers. Un-decoupled latch controllers is only able to store valid data in every second latch in FIFO. By using Semi or fully de-coupled latch controllers valid data can be stored in every latch, thus the number of handshake latches can be reduced by a factor of two. A fully de-coupled latch controller will be able to take advantage of the asynchronous delay element for improved performance.

Reliability of mutex The MTBF of the mutex component is unknown. The mutex is implemented solely using LUTs, thus the possibility of metastability failures may prove to be high. To investigate the reliability of the mutex experiments can be performed where the mutex is repeatedly put into metastability over a long time period. Also an analysis of the fairness of the mutex can be performed.

Delay elements As mentioned in the previous section a different design of the delay element can be considered.

Chapter 9

Conclusion

The purpose of the thesis has been to implement an asynchronous NoC prototype on an standard FPGA. The previous work about implementing asynchronous circuits on FPGAs is very limited thus a major part of the project have been to develop a general design flow for the implementation of asynchronous circuits on FPGAs.

A simple asynchronous best-effort NoC have been developed. The NoC consists of a router, a slave NA, and a master NA. The router is designed to work in a mesh topology and uses wormhole routing. Deadlock freedom is assured by using xy-routing and source routing is used. A packet can consist of an unlimited number of flits. To identify the beginning and end of a packet three flit types is used. The flit type is encoded by adding two additional request signals to the handshake channel. The NAs provides an OCP interface for the cores to connect to the network. Synchronization is handled using a simple two flip-flop synchronizer. The area usage of the router is 1295 LUTs and 330 latches where 29% of the LUTs is used by delay elements. The throughput of the router has been measured to be 43 MHz.

A small multi-processor prototype utilizing the asynchronous NoC have been developed. The prototype consists of three CPUs and three peripheral units which are connected by a 3x2 mesh topology. To assure that the system is free from message-dependant deadlocks a separate request and response net is used.

It has not been possible to fit a larger design on the FPGA. It has proven to be hard to reach a high utilization of the logic resources on the FPGA. It is suspected to be due to exhaustion of the routing resources. It is unclear if the implementation of the asynchronous NoC is due to these issues. There seems to be a general problem of reaching high utilization figures for the FPGA that has been used for the prototype. However, there are also indications that the implementation of the asynchronous NoC increases this problem.

The primary issues for implementing asynchronous circuits on FPGAs is the delay matching process. Also, unwanted optimizations by the design tools have been problematic. To optimally delay match a circuit the predictability of the delay of the delay element and the delay of the datapath must both be high. By using relative placement constraints in the design of the delay element satisfying predictability have been achieved. To reduce the fluctuations in the delay of the datapath it has been tried to create macros with locked placement of the design primitives. Due to unresolved problems with the design tools it has not been possible to create such macros. As a consequence it is needed to add extra delay during delay matching. It is not possible to turn off the logical optimizations performed by the design tools. They can be somewhat controlled by the use of different settings and constraints. For carefully designed circuits such as the circuits synthesized by Petrify is has the consequence that it is necessary to do the LUT mapping and placement manually. For other circuits it has not proven to be a large issue.

Bibliography

[1] Peter Alfke. "re: Xilinx is not specified minimum delay". http://groups.google.dk/group/comp.arch.fpga/msg/01d5ad08acadc337, 1996. Newsgroup: comp.arch.fpga.

[2] Peter Alfke. *XAPP094: Metastable Recovery in Virtex-II Pro FPGAs*. Xilinx, February 2005. Xilinx Application Note.

[3] Altera. Stratix iii fpgas vs. xilinx virtex-5 devices: Architecture and performance comparison, October 2007. White Paper.

[4] J. Bainbridge and S. Furber. Chain: a delay-insensitive chip area interconnect. *Micro, IEEE*, 22(5):16–23, Sep/Oct 2002.

[5] Tobias Bjerregaard. *The MANGO Clockless Network-on-Chip: Concepts and Implementation*. PhD thesis, Technical University of Denmark, 2005.

[6] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, Enric Pastor, and Alexandre Yakovlev. Petrify. http://www.lsi.upc.edu/ jordicf/petrify/. Version 4.2.

[7] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture – A Hardware/Software Approach*. Morgan Kaufmann, 1998. Chapter 10.

[8] Ran Ginosar. Course 048878 vlsi architectures – lecture slides, 2008. Lecture 3.

[9] Paul Glover and Steve Elzinga. Relationally placed macros. *TechXclusive*, August 2002.

[10] Esben Rosenlund Hansen and Anders Tranberg-Hansen. *Implementation of Asynchronous Circuits in FPGAs.* IMM/DTU, 2006.

[11] Knud Hansen and Guillaume Saoutieff. *02204 Course Project – VHDL Based Design Flow.* IMM/DTU, 2003.

[12] Andreas Hansson, Kees Goossens, and Andrei Radulescu. Avoiding message-dependent deadlock in network-based systems on chip. *VLSI Design, vol. 2007*, 2007.

[13] IST-2002. Aspida. http://www.ics.forth.gr/carv/async/demo/, 2004.

[14] Mads Havshøj Kristensen and Jon Neerup Lassen. *FPGA Implementation of an Asynchronous Arbiter.* IMM/DTU, 2007.

[15] Tue Strøjer Lyster and Morten Briand Thomsen. *Project in Asynchronous Systems.* IMM/DTU, 2004.

[16] OCP International Partnership. *Open Core Protocol Specification.* Release 2.0.

[17] Rasmus Grøndahl Olsen. Ocp based adapter for network-on-chip. Master's thesis, Technical University of Denmark, 2005.

[18] Open Verilog International. *Standard Delay Format Specification*, 3.0 edition, 1995.

[19] Opencores. http://www.opencores.org/.

[20] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits – A Design Perspective.* Prentice Hall, 2nd edition, 2003.

[21] Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart. *Asynchronous Circuits on FPGAs.* IMM/DTU, 2005.

[22] Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart. A noc-based soc executing a ray tracer, using synchronous multiprocessing, 2005. IMM, DTU. Polyteknisk Midtvejs Projekt.

[23] D. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar. An asynchronous router for multiple service levels networks on chip. *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pages 44–53, 14-16 March 2005.

[24] Jens Sparsø. *Asynchronous Circuit Design – A Tutorial.* Techinical University of Denmark, 2006.

[25] Jens Sparsø. Course 02204 design of asynchronous circuits – lecture slides, 2007. Lecture 8.

[26] Mikkel Bystrup Stensgaard. *Asynchronous Circuits in FPGA*. IMM/DTU, 2004.

[27] John F. Wakerly. *Digital Design – Principles and Practices*. Prentice Hall, 3rd edition, 2001.

[28] *Wishbone System-on-chip (SoC) Interconnection Architecture for Portable IP Cores*, 2002. Revision B.3.

[29] Xilinx. *Constraints Guide*. ISE 9.1i.

[30] Xilinx. *Floorplanner 9.2 Help*.

[31] Xilinx. *XAPP422: Creating RPMs Using 6.2i Floorplanner*, March 2004. Xilinx Application Note.

[32] Xilinx. Answer record: #23777. http://www.xilinx.com/support/answers/23777.htm, 2007.

[33] Xilinx. *ChipScope Pro Software and Cores User Guide*, 2007. Version 9.2.

[34] Xilinx. *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*, 2007.

[35] Xilinx. *Virtex-5 User Guide*, September 2007.

[36] Problems in using chipscope cdc file. http://forums.xilinx.com. Thread from the Xilinx Community Forums.

Appendix A

Appendices

A.1 Perl SDF script

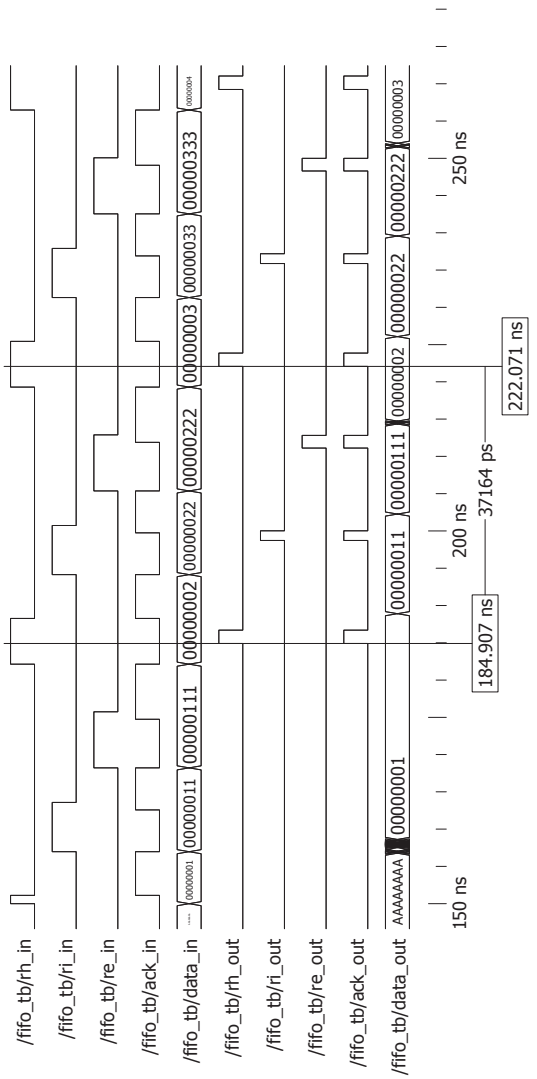
.

```
1  #!/usr/bin/perl
2
3  #Run script on sdf file to fix simulation problems with the mutex
4
5  if($#ARGV < 0 || $#ARGV > 1) {  # 1 or 2 arguments note: $#ARGV is the
6                                  #subscript of the last element in @ARGV
7    print "Usage: sdf.pl <input file> [<output file>]\n
8           If <output file> is not specified <input file>
9           will be used as output file.\n";
10   exit;
11 }
12
13 $input_file = @ARGV[0];
14 $output_file = $input_file;
15
16 #If <output file> specified
17 if($#ARGV = 1) {
18   $output_file = @ARGV[1];
19 }
20
21 print "Input file:\t$input_file\n";
22 print "Output file:\t$output_file\n";
23
24 #Open sdf file for reading
25 open(sdf_file, $input_file) || die "Could not open $filename
26                                    for reading: $!\n";
```

```
27
28  #Load file into string array
29  @lines = <sdf_file>;
30
31  #Close file
32  close sdf_file;
33
34  #Open sdf file for writing, reusing stdout
35  open(sdf_file, ">$output_file") || die "Could not open $filename
36                                          for writing: $!\n";
37
38  $instance = 0;
39  foreach $line (@lines) {
40
41      #Match Instance
42      if($line =~ /INSTANCE.+nand_1\)/) {
43        $instance = 1;
44        print $line;      #Debug
45      }
46
47      if($instance == 1) {
48
49        #Replace "PORT ADR4 (xxx)(xxx))" with "PORT ADR4 ( 0 )( 0 ))"
50        $line =~ s/PORT ADR4 \(.+\)\(.+\)\)/PORT ADR4 \( 0 \)\( 0 \)\)/;
51
52        #Replace "IOPATH ADR4 O (xxx)(xxx)" with "IOPATH ADR4 O ( 0 )( 0 ))
53        $line =~ s/(IOPATH ADR4 O) \(.+\)\(.+\)\)/$1 \( 0 \)\( 0 \)\)/;
54        #print $line; #Debug
55      }
56
57      #Match end of instance: "    )"
58      if($line =~ /^\s+\)/) {
59        $instance = 0;
60      }
61      print sdf_file $line;
62  }
63
64  close sdf_file;
65  print "Done!\n";
```

A.2 NoC Tests

A.2.1 FIFO



A.2.2 Input Port

Flit Type	Input	Expected Output
rh	0x00000001	0x00000004
ri	0x00000002	0x00000002
re	0x00000003	0x00000003
rh	0x40000011	0x00000045
ri	0x00000022	0x00000022
re	0x00000033	0x00000033
rh	0x80000111	0x00000446
ri	0x00000222	0x00000222
re	0x00000333	0x00000333
rh	0xc0001111	0x00004447
ri	0x00002222	0x00002222
re	0x00003333	0x00003333

/input_port_tb/reset
/input_port_tb/rh_in
/input_port_tb/ri_in
/input_port_tb/re_in
/input_port_tb/ack_in
/input_port_tb/data_in
/input_port_tb/a_rh_out
/input_port_tb/a_ri_out
/input_port_tb/a_re_out
/input_port_tb/a_ack_out
/input_port_tb/a_data_out
/input_port_tb/b_rh_out
/input_port_tb/b_ri_out
/input_port_tb/b_re_out
/input_port_tb/b_ack_out
/input_port_tb/b_data_out
/input_port_tb/c_rh_out
/input_port_tb/c_ri_out
/input_port_tb/c_re_out
/input_port_tb/c_ack_out
/input_port_tb/c_data_out
/input_port_tb/d_rh_out
/input_port_tb/d_ri_out
/input_port_tb/d_re_out
/input_port_tb/d_ack_out
/input_port_tb/d_data_out
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns

A.2.3 Output Port

Flit Type	Input	Expected Output
rh	0x00000001	0x00000001
ri	0x00000011	0x00000011
re	0x00000111	0x00000111
rh	0x00000002	0x00000002
ri	0x00000022	0x00000022
re	0x00000222	0x00000222
rh	0x00000003	0x00000003
ri	0x00000033	0x00000033
re	0x00000333	0x00000333
rh	0x00000004	0x00000004
ri	0x00000044	0x00000044
re	0x00000444	0x00000444

/output_port_tb/reset
/output_port_tb/a_rh
/output_port_tb/a_ri
/output_port_tb/a_re
/output_port_tb/a_ack
/output_port_tb/a_data
/output_port_tb/b_rh
/output_port_tb/b_ri
/output_port_tb/b_re
/output_port_tb/b_ack
/output_port_tb/b_data
/output_port_tb/c_rh
/output_port_tb/c_ri
/output_port_tb/c_re
/output_port_tb/c_ack
/output_port_tb/c_data
/output_port_tb/d_rh
/output_port_tb/d_ri
/output_port_tb/d_re
/output_port_tb/d_ack
/output_port_tb/d_data
/output_port_tb/output_rh
/output_port_tb/output_ri
/output_port_tb/output_re
/output_port_tb/output_ack
/output_port_tb/output_data
AAAAAAAA
00000001
00000011
00000111
00000002
00000022
00000222
00000003
00000033
00000333
00000004
00000044
00000444
00000000
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns

A.2.4 Router Simulation

Source	Dest.	Flit Type	Input	Expected Output
North	East	rh	0x4550	0x1541
		ri	0xAAA0	0xAAA0
		re	0x5550	0x5550
North	South	rh	0x8AA0	0x2A82
		ri	0x5550	0x5550
		re	0xAAA0	0xAAA0
North	West	rh	0xC550	0x1543
		ri	0xAAA0	0xAAA0
		re	0x5550	0x5550
North	Local	rh	0x0AA0	0x2A80
		ri	0x5550	0x5550
		re	0xAAA0	0xAAA0
East	North	rh	0x4551	0x1544
		ri	0xAAA1	0xAAA1
		re	0x5551	0x5551
East	South	rh	0x8AA1	0x2A86
		ri	0x5551	0x5551
		re	0xAAA1	0xAAA1
East	West	rh	0xC551	0x1547
		ri	0xAAA1	0xAAA1
		re	0x5551	0x5551
East	Local	rh	0x0AA1	0x2A85
		ri	0x5551	0x5551
		re	0xAAA1	0xAAA1
South	North	rh	0x4552	0x1548
		ri	0xAAA2	0xAAA2
		re	0x5552	0x5552
South	East	rh	0x8AA2	0x2A89
		ri	0x5552	0x5552
		re	0xAAA2	0xAAA2
South	West	rh	0xC552	0x154B
		ri	0xAAA2	0xAAA2
		re	0x5552	0x5552
South	Local	rh	0x0AA2	0x2A8A
		ri	0x5552	0x5552
		re	0xAAA2	0xAAA2

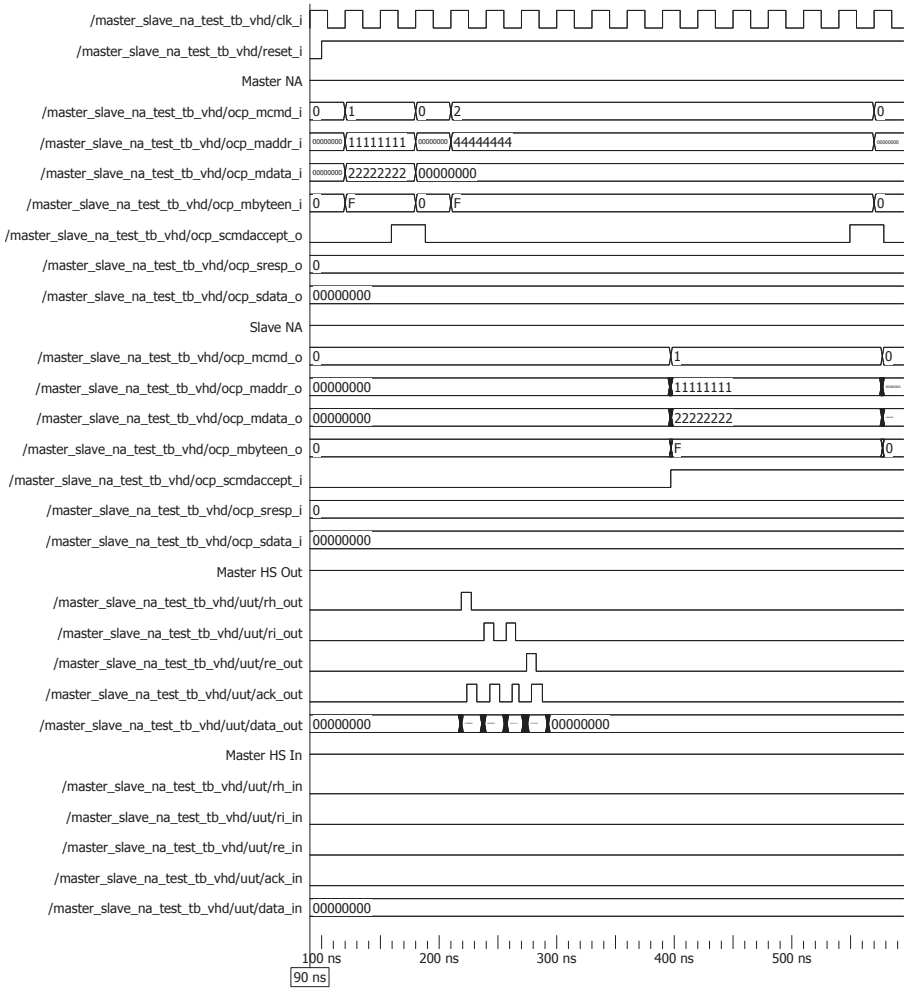
Source	Dest.	Flit Type	Input	Expected Output
West	North	rh	0x4553	0x154C
		ri	0xAAA3	0xAAA3
		re	0x5553	0x5553
West	East	rh	0x8AA3	0x2A8D
		ri	0x5553	0x5553
		re	0xAAA3	0xAAA3
West	South	rh	0xC553	0x154E
		ri	0xAAA3	0xAAA3
		re	0x5553	0x5553
West	Local	rh	0x0AA3	0x2A8F
		ri	0x5553	0x5553
		re	0xAAA3	0xAAA3
Local	North	rh	0x4554	0x1550
		ri	0xAAA4	0xAAA4
		re	0x5554	0x5554
Local	East	rh	0x8AA4	0x2A91
		ri	0x5554	0x5554
		re	0xAAA4	0xAAA4
Local	South	rh	0xC554	0x1552
		ri	0xAAA4	0xAAA4
		re	0x5554	0x5554
Local	Local	rh	0x0AA4	0x2A93
		ri	0x5554	0x5554
		re	0xAAA4	0xAAA4

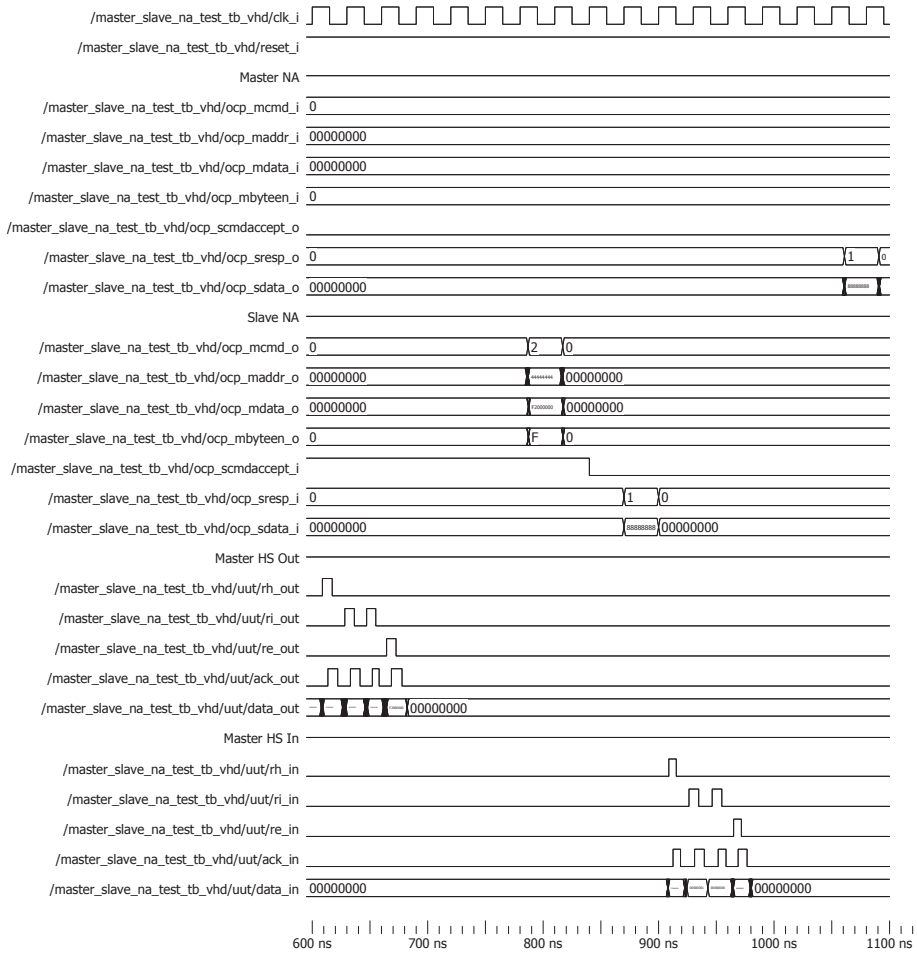
Input Ports
/be_router_boardtest_tb/uut/north_rh_in
/be_router_boardtest_tb/uut/north_ri_in
/be_router_boardtest_tb/uut/north_re_in
/be_router_boardtest_tb/uut/north_ack_in
/be_router_boardtest_tb/uut/north_data_in
/be_router_boardtest_tb/uut/east_rh_in
/be_router_boardtest_tb/uut/east_ri_in
/be_router_boardtest_tb/uut/east_re_in
/be_router_boardtest_tb/uut/east_ack_in
/be_router_boardtest_tb/uut/east_data_in
/be_router_boardtest_tb/uut/south_rh_in
/be_router_boardtest_tb/uut/south_ri_in
/be_router_boardtest_tb/uut/south_re_in
/be_router_boardtest_tb/uut/south_ack_in
/be_router_boardtest_tb/uut/south_data_in
/be_router_boardtest_tb/uut/west_rh_in
/be_router_boardtest_tb/uut/west_ri_in
/be_router_boardtest_tb/uut/west_re_in
/be_router_boardtest_tb/uut/west_ack_in
/be_router_boardtest_tb/uut/west_data_in
/be_router_boardtest_tb/uut/local_rh_in
/be_router_boardtest_tb/uut/local_ri_in
/be_router_boardtest_tb/uut/local_re_in
/be_router_boardtest_tb/uut/local_ack_in
/be_router_boardtest_tb/uut/local_data_in
0000 4550 AAA0 5550 8AA0 5550 AAA0 C550 AAA0 5550 0AA0 5550 AAA0 4550 AAA0 5550 8AA0
0000 0551 AAA1 5551 8AA1 5551 AAA1 C551 AAA1 5551 4AA1 5551 AAA1 0551 AAA1 5551 8AA1 5551 AAA1 C551
0000 0552 AAA2 5552 4AA2 5552 AAA2 C552 AAA2 5552 8AA2
0000 0553 AAA3 5553 4AA3 5553 AAA3 8553 AAA3 5553 CAA3 AAA3 5553 0553
0000 0554 AAA4 5554 4AA4 5554 AAA4 8554 AAA4 5554 CAA4 5554 AAA4 0554 AAA4 5554 4AA4
200 ns
300 ns
400 ns
500 ns
600 ns

Outputs Ports
/be_router_boardtest_tb/uut/north_rh_out
/be_router_boardtest_tb/uut/north_ri_out
/be_router_boardtest_tb/uut/north_re_out
/be_router_boardtest_tb/uut/north_ack_out
/be_router_boardtest_tb/uut/north_data_out
/be_router_boardtest_tb/uut/east_rh_out
/be_router_boardtest_tb/uut/east_ri_out
/be_router_boardtest_tb/uut/east_re_out
/be_router_boardtest_tb/uut/east_ack_out
/be_router_boardtest_tb/uut/east_data_out
/be_router_boardtest_tb/uut/south_rh_out
/be_router_boardtest_tb/uut/south_ri_out
/be_router_boardtest_tb/uut/south_re_out
/be_router_boardtest_tb/uut/south_ack_out
/be_router_boardtest_tb/uut/south_data_out
/be_router_boardtest_tb/uut/west_rh_out
/be_router_boardtest_tb/uut/west_ri_out
/be_router_boardtest_tb/uut/west_re_out
/be_router_boardtest_tb/uut/west_ack_out
/be_router_boardtest_tb/uut/west_data_out
/be_router_boardtest_tb/uut/local_rh_out
/be_router_boardtest_tb/uut/local_ri_out
/be_router_boardtest_tb/uut/local_re_out
/be_router_boardtest_tb/uut/local_ack_out
/be_router_boardtest_tb/uut/local_data_out
200 ns
300 ns
400 ns
500 ns
600 ns
176.48 ns
69928 ps
246.408 ns

A.2.5 Router On-Board

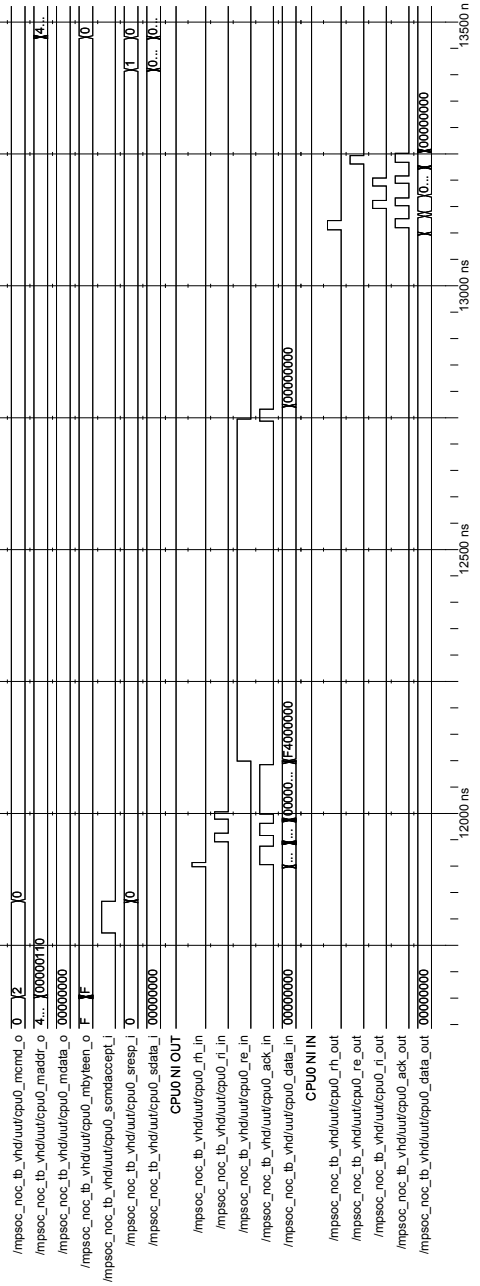
A.2.6 Network Adapter



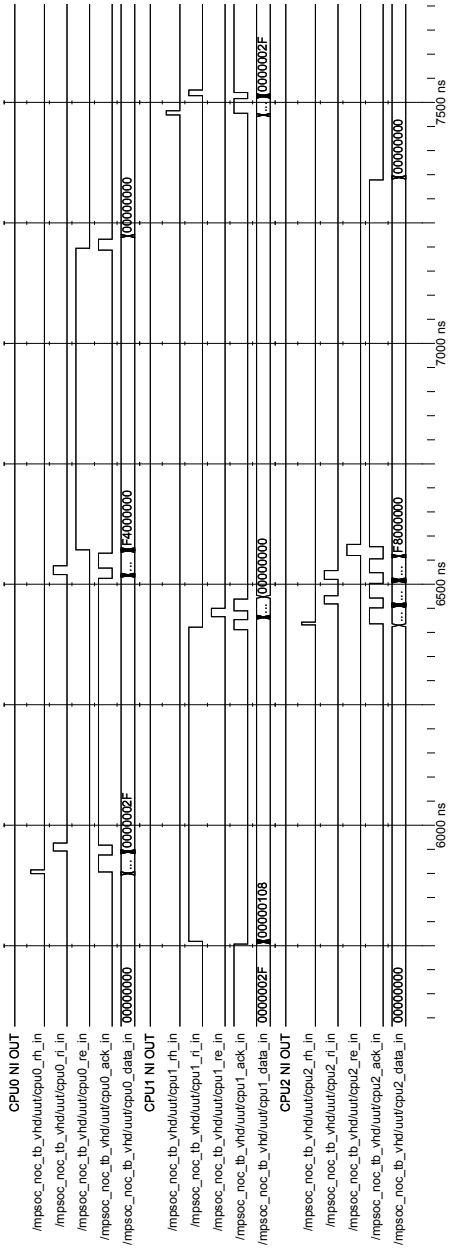

/master_slave_na_test_tb_vhd/clk_i
/master_slave_na_test_tb_vhd/reset_i
Master NA
/master_slave_na_test_tb_vhd/ocp_mcmd_i
/master_slave_na_test_tb_vhd/ocp_maddr_i
/master_slave_na_test_tb_vhd/ocp_mdata_i
/master_slave_na_test_tb_vhd/ocp_mbyteen_i
/master_slave_na_test_tb_vhd/ocp_scmdaccept_o
/master_slave_na_test_tb_vhd/ocp_sresp_o
/master_slave_na_test_tb_vhd/ocp_sdata_o
Slave NA
/master_slave_na_test_tb_vhd/ocp_mcmd_o
/master_slave_na_test_tb_vhd/ocp_maddr_o
/master_slave_na_test_tb_vhd/ocp_mdata_o
/master_slave_na_test_tb_vhd/ocp_mbyteen_o
/master_slave_na_test_tb_vhd/ocp_scmdaccept_i
/master_slave_na_test_tb_vhd/ocp_sresp_i
/master_slave_na_test_tb_vhd/ocp_sdata_i
Master HS Out
/master_slave_na_test_tb_vhd/uut/rh_out
/master_slave_na_test_tb_vhd/uut/ri_out
/master_slave_na_test_tb_vhd/uut/re_out
/master_slave_na_test_tb_vhd/uut/ack_out
/master_slave_na_test_tb_vhd/uut/data_out
Master HS In
/master_slave_na_test_tb_vhd/uut/rh_in
/master_slave_na_test_tb_vhd/uut/ri_in
/master_slave_na_test_tb_vhd/uut/re_in
/master_slave_na_test_tb_vhd/uut/ack_in
/master_slave_na_test_tb_vhd/uut/data_in
00000000
600 ns
700 ns
800 ns
900 ns
1000 ns
1100 ns

A.3 MPSoC Tests

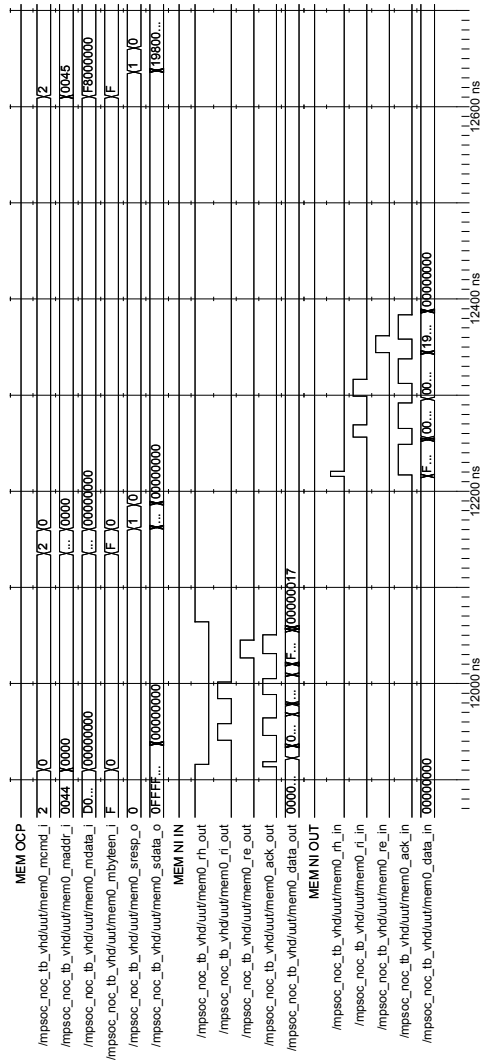
A.3.1 Case 1


CPU0 OCP
/mpsoc_noc_tb_vhd/uut/cpu0_mcmd_o
/mpsoc_noc_tb_vhd/uut/cpu0_maddr_o
/mpsoc_noc_tb_vhd/uut/cpu0_mdata_o
/mpsoc_noc_tb_vhd/uut/cpu0_mbyteen_o
/mpsoc_noc_tb_vhd/uut/cpu0_scmdaccept_i
/mpsoc_noc_tb_vhd/uut/cpu0_sresp_i
/mpsoc_noc_tb_vhd/uut/cpu0_sdata_i
CPU0 NI OUT
/mpsoc_noc_tb_vhd/uut/cpu0_rh_in
/mpsoc_noc_tb_vhd/uut/cpu0_ri_in
/mpsoc_noc_tb_vhd/uut/cpu0_re_in
/mpsoc_noc_tb_vhd/uut/cpu0_ack_in
/mpsoc_noc_tb_vhd/uut/cpu0_data_in
CPU0 NI IN
/mpsoc_noc_tb_vhd/uut/cpu0_rh_out
/mpsoc_noc_tb_vhd/uut/cpu0_re_out
/mpsoc_noc_tb_vhd/uut/cpu0_ri_out
/mpsoc_noc_tb_vhd/uut/cpu0_ack_out
/mpsoc_noc_tb_vhd/uut/cpu0_data_out
12000 ns
12500 ns
13000 ns
13500 n

A.3.2 Case 2


CPU0 NI OUT
/mpsoc_noc_tb_vhd/uut/cpu0_rh_in
/mpsoc_noc_tb_vhd/uut/cpu0_ri_in
/mpsoc_noc_tb_vhd/uut/cpu0_re_in
/mpsoc_noc_tb_vhd/uut/cpu0_ack_in
/mpsoc_noc_tb_vhd/uut/cpu0_data_in
CPU1 NI OUT
/mpsoc_noc_tb_vhd/uut/cpu1_rh_in
/mpsoc_noc_tb_vhd/uut/cpu1_ri_in
/mpsoc_noc_tb_vhd/uut/cpu1_re_in
/mpsoc_noc_tb_vhd/uut/cpu1_ack_in
/mpsoc_noc_tb_vhd/uut/cpu1_data_in
CPU2 NI OUT
/mpsoc_noc_tb_vhd/uut/cpu2_rh_in
/mpsoc_noc_tb_vhd/uut/cpu2_ri_in
/mpsoc_noc_tb_vhd/uut/cpu2_re_in
/mpsoc_noc_tb_vhd/uut/cpu2_ack_in
/mpsoc_noc_tb_vhd/uut/cpu2_data_in
00000000
10000002F
F4000000
00000000
0000002F
00000108
00000000
10000002F
00000000
F8000000
00000000
6000 ns
6500 ns
7000 ns
7500 ns

A.3.3 Case 3


MEM OCP
/mpsoc_noc_tb_vhd/uut/mem0_mcmd_i
/mpsoc_noc_tb_vhd/uut/mem0_maddr_i
/mpsoc_noc_tb_vhd/uut/mem0_mdata_i
/mpsoc_noc_tb_vhd/uut/mem0_mbyteen_i
/mpsoc_noc_tb_vhd/uut/mem0_sresp_o
/mpsoc_noc_tb_vhd/uut/mem0_sdata_o
MEM NI IN
/mpsoc_noc_tb_vhd/uut/mem0_rh_out
/mpsoc_noc_tb_vhd/uut/mem0_ri_out
/mpsoc_noc_tb_vhd/uut/mem0_re_out
/mpsoc_noc_tb_vhd/uut/mem0_ack_out
/mpsoc_noc_tb_vhd/uut/mem0_data_out
MEM NI OUT
/mpsoc_noc_tb_vhd/uut/mem0_rh_in
/mpsoc_noc_tb_vhd/uut/mem0_ri_in
/mpsoc_noc_tb_vhd/uut/mem0_re_in
/mpsoc_noc_tb_vhd/uut/mem0_ack_in
/mpsoc_noc_tb_vhd/uut/mem0_data_in
12000 ns
12200 ns
12400 ns
12600 ns

A.4 RPM Forum Post

```
Hi,

I have some trouble when I try to generate a RPM using the floorplanner
tool in Xilinx. I'm trying to make a RPM from a quite large design but
cannot get it to work. Now I'm trying with a simplified sub-module of
my design (see below).
The merge_test entity is a simple demux with 3 select signals for each
data signal.

When I synthesize and implement in Xilinx ISE I use the standard
settings except that I unchecks the insertion of I/O buffers and
trimming of unconnected signals.

After PAR I load the design into floorplanner and selects
floorplan->replace all with placement.
I get the first problem after executing "replace all with placement".
Two gates in the Design Hierarchy window is still unplaced! I have to
place them manually to get them included in the RPM.

Next issue:
When I count the number of LUTs showing up in floorplanner I only get
18 LUTs including the two unplaced gates. Two LUTs are missing!
So I loads the design into FPGA Editor and I'm able to locate all 20
LUTs. The four problematic LUTs are the 4 3-input OR-gates for or-ing
the select signals together.
The four problematic are all marked as "Route Through"s in FPGA Editor.

What are a route through?
How do I get all LUTs to show up and get placed in floorplanner so
I'm able to generate an RPM of the design?

The target is a Xilinx Virtex-5 FPGA.

Any help is appreciated.

Kind Regards
Jon Neerup Lassen

------ BEGIN VHDL -------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity merge_test is
    port(

        a0,a1,a2 : in  std_logic;
      b0,b1,b2 : in  std_logic;
      c0,c1,c2 : in  std_logic;
      d0,d1,d2 : in  std_logic;

      a_data     : in  std_logic_vector(7 downto 0);
      b_data     : in  std_logic_vector(7 downto 0);
      c_data     : in  std_logic_vector(7 downto 0);
      d_data     : in  std_logic_vector(7 downto 0);

      z_data     : out std_logic_vector(7 downto 0)

    );
```

```
end merge_test;

architecture arch of merge_test is

    signal a,b,c,d : std_logic;

begin

    data_demux : process(a0,a1,a2,b0,b1,b2,c0,c1,c2,d0,d1,d2,
                         a_data,b_data,c_data,d_data)
    begin
        if      (a0 or a1 or a2) = '1' then
            z_data <= a_data;
        elsif (b0 or b1 or b2) = '1' then
            z_data <= b_data;
        elsif (c0 or c1 or c2) = '1' then
            z_data <= c_data;
        elsif (d0 or d1 or d2) = '1' then
            z_data <= d_data;
        else
            z_data <= (others => '0');
        end if;
    end process;

end arch;

----- END VHDL -----
```

A.5 VHDL Code

A.5.1 Async Design Elements

A.5.1.1 as_bd_4p_delay.vhd

```
1  ------------------------------------------------------------------------
2  --  Title        : as_bd_4p_delay.vhd
3  --               :
4  --  Developer    : Mikkel Stensgaard  -- mikkel@stensgaard.org
5  --               : Student: s001434
6  --               :
7  --  Version 1.1 : Anders Tranberg-Hansen, s011509@student.dtu.dk
8  --  by           : Esben Rosenlund Hansen, s011579@student.dtu.dk
9  --               :
10 --  Version 1.2 :
11 --  by           : Jon Neerup Lassen, s020310@student.dtu.dk
12 --               : DTU, Technical University of Denmark
13 --               :
14 --  Revision     : 1.0      ??-??-04     Initial version
15 --               : 1.1      05-01-06     "Niceified" version and further a
16 --               :                       wrong reference to lut in unisim
17 --               :                       library fixed. Added "after"-clause
18 --               :                       which allows a delay in simulations.
19 --               :                       The delay element can now be proper
20 --               :                       simulated.
21 --               : 1.2      08-11-07     Using rloc constraints to control
22 --               :                       placement of luts.
23 --               :
```

```
24  ----------------------------------------------------------------------------
25  library ieee;
26  use ieee.std_logic_1164.all;
27
28  library unisim;
29  use unisim.vcomponents.lut2;
30  use unisim.vcomponents.lut1;
31
32  entity as_bd_4p_delay is
33    generic(
34      size : natural range 1 to 30 := 10  -- Delay size
35    );
36    port (
37      d : in  std_logic;     -- Data in
38      z : out std_logic      -- Data out
39    );
40  end as_bd_4p_delay;
41
42  architecture lut of as_bd_4p_delay is
43
44    component lut2
45      generic (
46        init : bit_vector := X"4"
47      );
48      port (
49        o  : out std_ulogic;
50        i0 : in  std_ulogic;
51        i1 : in  std_ulogic
52      );
53    end component;
54
55    ------------------------------------------------
56    -- Internal signals.
57    ------------------------------------------------
58    signal s_connect : std_logic_vector(size downto 0);
59    --signal d_inv,o_first   : std_logic;
60
61    ------------------------------------------------
62    --  Synthesis attributes - we don't want the
63    --  synthesizer to optimize the delay-chain.
64    ------------------------------------------------
65    attribute keep : string;
66    attribute keep of s_connect : signal is "true"; --d_inv
67
68    attribute rloc : string;
69
70  begin
71
72    s_connect(0) <= d;
73
74    ------------------------------------------------
75    -- Create a riple-chain of luts (and gates).
76    ------------------------------------------------
77    lut_chain : for index in 0 to (size-1) generate
78
79      signal o : std_logic;
80
81      type y_placement is array (integer range 0 to 29) of integer;
82      constant y_val : y_placement :=
            (0,1,0,1,0,1,0,1,2,3,2,3,2,3,2,3,4,5,4,5,4,5,4,5,6,7,6,7,6,7);
83
84      attribute rloc of delay_lut : label is "X0Y" & integer'image(y_val(index)
            );
85
86      begin
```

```
87        delay_lut: lut2
88        generic map(
89          init => "1000" -- And truth-table.
90        )
91        port map(
92          I1 => d,
93          I0 => s_connect(index),
94          O  => o
95
96        );
97        -- Simulate delay of 1 ns.
98        s_connect(index+1) <= o after 1 ns;
99
100     end generate lut_chain;
101
102     -------------------------------------------------
103     -- Connect the output of delay element
104     -------------------------------------------------
105     z <= s_connect(size-1);
106   end lut;
```

A.5.1.2 as_bd_4p_c2.vhd

```
1   --------------------------------------------------------------------------
2   --  Title       : AS_C2
3   --              :
4   --  Developer   : Mikkel Stensgaard  -- mikkel@stensgaard.org
5   --              : Student: s001434
6   --              :
7   --  Version 1.1 : Anders Tranberg-Hansen, s011509@student.dtu.dk
8   --  by          : Esben Rosenlund Hansen, s011579@student.dtu.dk
9   --              :
10  --              : DTU, Technical University of Denmark
11  --              :
12  --  Revision    : 1.0     ??-??-04    Initial version
13  --              : 1.1     05-01-06    "Niceified" version.
14  --              :
15  --------------------------------------------------------------------------
16  library ieee;
17  use ieee.std_logic_1164.all;
18
19  library unisim;
20  use unisim.vcomponents.lut4_l;
21
22  entity as_bd_4p_c2 is
23    generic(
24      reset_value : bit := '0'  -- Reset value of output
25    );
26    port (
27      reset : in  std_logic;    -- Reset (Active low)
28      a     : in  std_logic;    -- Input A
29      b     : in  std_logic;    -- Input B
30      z     : out std_logic     -- Output Z
31    );
32  end as_bd_4p_c2;
33
34  architecture lut of as_bd_4p_c2 is
35    -------------------------------------------------
36    -- Create the reset-vector as a constant
37    -- using the generic "reset_value".
38    -------------------------------------------------
39    constant rv : bit := reset_value;
40    constant reset_vector : bit_vector(7 downto 0) := rv&rv&rv&rv&rv&rv&rv&rv;
```

```
41
42    ------------------------------------------------
43    -- Internal signals
44    ------------------------------------------------
45    signal s_out : std_logic;
46
47    attribute keep : string;
48    attribute keep of s_out : signal is "true";
49
50  begin
51    ------------------------------------------------
52    -- The logical equation defining the C element is:
53    --
54    -- out_MC = reset AND ((ina AND inb) OR
55    --           (out_MC AND (in_a OR in_b)))
56    ------------------------------------------------
57    c_element: lut4_1
58    generic map (
59      init => "11101000" & reset_vector
60    )
61    port map (
62      i0 => a,
63      i1 => b,
64      i2 => s_out,
65      i3 => reset,
66      lo => s_out
67    );
68
69    ------------------------------------------------
70    -- Connect the internal signals to the outputs.
71    ------------------------------------------------
72    z <= s_out after 1 ns;
73
74  end lut;
```

A.5.1.3 as_bd_4p_mutex.vhd

```
1   ------------------------------------------------------------------------------
2   --  Title       : as_bd_4p_mutex.vhd
3   --              :
4   --  Developer   : Mads Kristensen,  s061732@student.dtu.dk
5   --              : Jon Lassen,       s020310@student.dtu.dk
6   --              :
7   --              : DTU, Technical University of Denmark
8   --              :
9   --  Revision    : 1.0     22-05-07    Initial version
10  --              : 1.1     15-12-07    Changed rlocs to fit a Virtex5
11  --              :
12  ------------------------------------------------------------------------------
13  library ieee;
14  use ieee.std_logic_1164.all;
15  use ieee.math_real.all; -- for UNIFORM
16
17  library unisim;
18  use unisim.vcomponents.lut3_l;
19  use unisim.vcomponents.lut2_l;
20  use unisim.vcomponents.lut3;
21  use unisim.vcomponents.lut2;
22  use unisim.vcomponents.lut4_l;
23
24  --library as_fpga;
25  --use as_fpga.as_bd_4p.all;
26
```

```
27  entity as_bd_4p_mutex is
28    generic(
29      reset_value : bit := '0'; -- Reset value of output
30      --Only used for the random number generator for the behavioral arch.
31      seed1 : positive := 1;
32      seed2 : positive := 1
33    );
34    port (
35       reset  : in  std_logic;     -- Reset (Active low)
36      r1,r2   : in  std_logic;     -- in
37      g1,g2   : out std_logic      -- mutex out
38    );
39
40  end as_bd_4p_mutex;
41
42  -- Architecture for implementation
43  architecture gate of as_bd_4p_mutex is
44  ---------------------------------------------------
45  -- Create the reset-vector as a constant
46  -- using the generic "reset_value".
47  ---------------------------------------------------
48     constant rv : bit := reset_value;
49     constant reset_vector : bit_vector(3 downto 0) := rv&rv&rv&rv;
50
51  ---------------------------------------------------
52  -- Internal signals
53  ---------------------------------------------------
54     signal o1,o2,o2_delayed : std_logic;
55  ---------------------------------------------------
56  -- Xilinx contraints
57  ---------------------------------------------------
58
59    attribute rloc : string;
60
61    --for virtex5
62    attribute rloc of nand_1:label is "X0Y0";
63    attribute rloc of nand_2:label is "X0Y0";
64    attribute rloc of  and_1:label is "X1Y0";
65    attribute rloc of  and_2:label is "X1Y0";
66
67    --for virtex2
68  --  attribute rloc of nand_1:label is "X0Y1";
69  --  attribute rloc of nand_2:label is "X0Y1";
70  --  attribute rloc of  and_1:label is "X0Y0";
71  --  attribute rloc of  and_2:label is "X0Y0";
72
73  begin
74
75    -- 3-Bit Look-Up-Table with Local Output
76    -- performs logic function: not(i0 AND i1 AND not(i2))
77     nand_1: lut3_l
78     generic map (
79         init => "0111" & "1111"
80      )
81     port map (
82         i0 => r1,
83         i1 => o2_delayed,
84         i2 => reset,
85         lo => o1
86     );
87
88    -- 3-Bit Look-Up-Table with Local Output
89    -- performs logic function: not(i0 AND i1 AND not(i2))
90    nand_2: lut3_l
91    generic map (
```

```
92      init =>  "0111" & "1111"
93    )
94    port map (
95      i0 => r2,
96      i1 => o1,
97      i2 => reset,
98      lo => o2
99    );
100
101   --Apply inertial delay to allow simulation. Note this makes the mutex
          unfair! Is ignored when synthesized.
102   o2_delayed <= o2 after 1 ns;
103
104    -- 3-Bit Look-Up-Table with normal Output
105    -- performs logic function: not(i0) AND i2 AND not(i2)
106    and_1: lut3
107       generic map (
108          init =>  "0100"& reset_vector
109       )
110       port map (
111          i0 => o1,
112          i1 => o2,
113          i2 => reset,
114          o => g1
115       );
116
117   -- 3-Bit Look-Up-Table with normal Output
118   -- performs logic function: not(i0) AND i2 AND not(i2)
119   and_2: lut3
120   generic map (
121     init => "0100" &  reset_vector
122   )
123   port map (
124     i0 => o2,
125     i1 => o1,
126     i2 => reset,
127     o => g2
128   );
129
130 end architecture gate;
131
132 -- Architecture for behavioral simulation
133 architecture behaviour of as_bd_4p_mutex is
134   signal dummy : std_logic;
135 begin
136
137   mutex: process(r1,r2,reset)
138     variable rand : real;
139     variable v_seed1 : positive;
140     variable v_seed2 : positive;
141   begin
142
143     if reset = '0' then
144       g1 <= '0';
145       g2 <= '0';
146       v_seed1 := seed1; --initialize seed values
147       v_seed2 := seed2;
148     elsif (r1'event and r1='1') and (r2'event and r2='1') then
149       UNIFORM(v_seed1, v_seed2, rand); --get random value between 0 and (
              almost) 1
150       if rand < 0.5 then
151         g1 <= '1';
152         g2 <= '0';
153       else
154         g1 <= '0';
```

```
155           g2 <= '1';
156         end if;
157       elsif (r1='1') and (r2='0') then
158         g1 <= '1';
159         g2 <= '0';
160       elsif (r1='0') and (r2='1') then
161         g1 <= '0';
162         g2 <= '1';
163       elsif (r1='1') and (r2='1') then
164         dummy <= dummy; --(NOP)
165       else
166         g1 <= '0';
167         g2 <= '0';
168       end if;
169     end process mutex;
170
171   end architecture behaviour;
```

A.5.1.4 synchronizer.vhd

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  library UNISIM;
 7  use UNISIM.VComponents.fdc;
 8
 9  entity synchronizer is
10    port(
11      clk_in    : in  std_logic;
12      reset_i   : in  std_logic;
13      async_in  : in  std_logic;
14      sync_out  : out std_logic
15    );
16  end synchronizer;
17
18  architecture Behavioral of synchronizer is
19
20    component FDC
21      generic (INIT : bit:= '0');
22    port (
23      Q : out STD_ULOGIC;
24      C : in STD_ULOGIC;
25      CLR : in STD_ULOGIC;
26      D : in STD_ULOGIC
27    );
28    end component;
29
30    signal ff0_out, reset_inv : std_logic;
31
32    attribute rloc : string;
33    attribute rloc of ff0 : label is "X0Y0";
34    attribute rloc of ff1 : label is "X0Y0";
35
36  begin
37
38    reset_inv <= not reset_i;
39
40  -- Two ff synchronizer
41
42    ff0 : fdc
43    port map (
```

```
44      Q   => ff0_out,
45      C   => clk_in,
46      CLR => reset_inv,
47      D   => async_in
48    );
49
50    ff1 : fdc
51    port map (
52      Q   => sync_out,
53      C   => clk_in,
54      CLR => reset_inv,
55      D   => ff0_out
56    );
57
58  end Behavioral;
```

A.5.2 Router

A.5.2.1 be_router.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   entity be_router is
8       port(
9         reset : in std_logic;
10        -- Input ports --
11        north_rh_in   : in  std_logic;
12        north_ri_in   : in  std_logic;
13        north_re_in   : in  std_logic;
14        north_ack_in  : out std_logic;
15        north_data_in : in  flit_data;
16
17        west_rh_in    : in  std_logic;
18        west_ri_in    : in  std_logic;
19        west_re_in    : in  std_logic;
20        west_ack_in   : out std_logic;
21        west_data_in  : in  flit_data;
22
23        south_rh_in   : in  std_logic;
24        south_ri_in   : in  std_logic;
25        south_re_in   : in  std_logic;
26        south_ack_in  : out std_logic;
27        south_data_in : in  flit_data;
28
29        east_rh_in    : in  std_logic;
30        east_ri_in    : in  std_logic;
31        east_re_in    : in  std_logic;
32        east_ack_in   : out std_logic;
33        east_data_in  : in  flit_data;
34
35        local_rh_in   : in  std_logic;
36        local_ri_in   : in  std_logic;
37        local_re_in   : in  std_logic;
38        local_ack_in  : out std_logic;
39        local_data_in : in  flit_data;
40
41        -- Output ports --
```

```
42        north_rh_out    : out std_logic;
43        north_ri_out    : out std_logic;
44        north_re_out    : out std_logic;
45        north_ack_out   : in  std_logic;
46        north_data_out  : out flit_data;
47
48        west_rh_out     : out std_logic;
49        west_ri_out     : out std_logic;
50        west_re_out     : out std_logic;
51        west_ack_out    : in  std_logic;
52        west_data_out   : out flit_data;
53
54        south_rh_out    : out std_logic;
55        south_ri_out    : out std_logic;
56        south_re_out    : out std_logic;
57        south_ack_out   : in  std_logic;
58        south_data_out  : out flit_data;
59
60        east_rh_out     : out std_logic;
61        east_ri_out     : out std_logic;
62        east_re_out     : out std_logic;
63        east_ack_out    : in  std_logic;
64        east_data_out   : out flit_data;
65
66        local_rh_out    : out std_logic;
67        local_ri_out    : out std_logic;
68        local_re_out    : out std_logic;
69        local_ack_out   : in  std_logic;
70        local_data_out  : out flit_data
71      );
72
73  end be_router;
74
75  architecture struct of be_router is
76
77  -- Component declarations --
78    component input_port is
79      port (
80        reset       : in std_logic;
81        -- Input channel --
82        rh_in       : in  std_logic;
83        ri_in       : in  std_logic;
84        re_in       : in  std_logic;
85        ack_in      : out std_logic;
86        data_in     : in  flit_data;
87        -- Output channel A --
88        a_rh_out    : out std_logic;
89        a_ri_out    : out std_logic;
90        a_re_out    : out std_logic;
91        a_ack_out   : in  std_logic;
92        a_data_out  : out flit_data;
93        -- Output channel B --
94        b_rh_out    : out std_logic;
95        b_ri_out    : out std_logic;
96        b_re_out    : out std_logic;
97        b_ack_out   : in  std_logic;
98        b_data_out  : out flit_data;
99        -- Output channel C --
100       c_rh_out    : out std_logic;
101       c_ri_out    : out std_logic;
102       c_re_out    : out std_logic;
103       c_ack_out   : in  std_logic;
104       c_data_out  : out flit_data;
105       -- Output channel LOCAL --
106       d_rh_out    : out std_logic;
```

```
107         d_ri_out    : out std_logic;
108         d_re_out    : out std_logic;
109         d_ack_out   : in  std_logic;
110         d_data_out  : out flit_data
111       );
112     end component;
113
114     component output_port is
115       port(
116         reset   : in  std_logic;  --Reset (active low)
117
118         a_rh    : in  std_logic;  --Handshake signals for input a
119         a_ri    : in  std_logic;
120         a_re    : in  std_logic;
121         a_ack   : out std_logic;
122
123         b_rh    : in  std_logic;  --Handshake signals for input b
124         b_ri    : in  std_logic;
125         b_re    : in  std_logic;
126         b_ack   : out std_logic;
127
128         c_rh    : in  std_logic;  --Handshake signals for input c
129         c_ri    : in  std_logic;
130         c_re    : in  std_logic;
131         c_ack   : out std_logic;
132
133         d_rh    : in  std_logic;  --Handshake signals for IP input
134         d_ri    : in  std_logic;
135         d_re    : in  std_logic;
136         d_ack   : out std_logic;
137
138         output_rh   : out std_logic;  --Handshake signals for output port
139         output_ri   : out std_logic;
140         output_re   : out std_logic;
141         output_ack  : in  std_logic;
142
143         a_data      : in  flit_data;  -- Data signals
144         b_data      : in  flit_data;
145         c_data      : in  flit_data;
146         d_data  : in  flit_data;
147         output_data : out flit_data
148       );
149     end component;
150
151     component fifo is
152     generic(
153       depth : positive := 1
154     );
155     port(
156       reset  : in std_logic;
157       rh_in : in  STD_LOGIC;
158       ri_in : in  STD_LOGIC;
159       re_in : in  STD_LOGIC;
160       ack_in : out  STD_LOGIC;
161       data_in : in  flit_data;
162       rh_out : out  STD_LOGIC;
163       ri_out : out  STD_LOGIC;
164       re_out : out  STD_LOGIC;
165       ack_out : in  STD_LOGIC;
166       data_out : out  flit_data
167     );
168     end component;
169
170     -- Internal signals --
171
```

```
172   -- crossbar handshake signals
173   signal north_east_rh,   north_east_ri,  north_east_re,  north_east_ack,
174          north_south_rh,  north_south_ri, north_south_re, north_south_ack,
175          north_west_rh,   north_west_ri,  north_west_re,  north_west_ack,
176          north_local_rh,  north_local_ri, north_local_re, north_local_ack,
177          east_south_rh,   east_south_ri,  east_south_re,  east_south_ack,
178          east_west_rh,    east_west_ri,   east_west_re,   east_west_ack,
179          east_north_rh,   east_north_ri,  east_north_re,  east_north_ack,
180          east_local_rh,   east_local_ri,  east_local_re,  east_local_ack,
181          south_west_rh,   south_west_ri,  south_west_re,  south_west_ack,
182          south_north_rh,  south_north_ri, south_north_re, south_north_ack,
183          south_east_rh,   south_east_ri,  south_east_re,  south_east_ack,
184          south_local_rh,  south_local_ri, south_local_re, south_local_ack,
185          west_north_rh,   west_north_ri,  west_north_re,  west_north_ack,
186          west_east_rh,    west_east_ri,   west_east_re,   west_east_ack,
187          west_south_rh,   west_south_ri,  west_south_re,  west_south_ack,
188          west_local_rh,   west_local_ri,  west_local_re,  west_local_ack,
189          local_north_rh,  local_north_ri, local_north_re, local_north_ack,
190          local_east_rh,   local_east_ri,  local_east_re,  local_east_ack,
191          local_south_rh,  local_south_ri, local_south_re, local_south_ack,
192          local_west_rh,   local_west_ri,  local_west_re,  local_west_ack
                    : std_logic;
193
194   -- crossbar data signals
195   signal north_east_data,   north_south_data, north_west_data,
          north_local_data,
196          east_south_data,   east_west_data,   east_north_data,
                east_local_data,
197          south_west_data,   south_north_data, south_east_data,
                south_local_data,
198          west_north_data,   west_east_data,   west_south_data,
                west_local_data,
199          local_north_data,  local_east_data,  local_south_data,
                local_west_data   : flit_data;
200
201   --input fifo signals
202   signal north_input_fifo_rh, north_input_fifo_ri, north_input_fifo_re,
          north_input_fifo_ack,
203          east_input_fifo_rh,  east_input_fifo_ri,  east_input_fifo_re,
                east_input_fifo_ack,
204          south_input_fifo_rh, south_input_fifo_ri, south_input_fifo_re,
                south_input_fifo_ack,
205          west_input_fifo_rh,  west_input_fifo_ri,  west_input_fifo_re,
                west_input_fifo_ack,
206          local_input_fifo_rh, local_input_fifo_ri, local_input_fifo_re,
                local_input_fifo_ack  : std_logic;
207
208   -- input fifo data signals
209   signal north_input_fifo_data, east_input_fifo_data, south_input_fifo_data,
210          west_input_fifo_data,  local_input_fifo_data
                    : flit_data;
211
212
213   --output fifo signals
214   signal north_output_fifo_rh, north_output_fifo_ri, north_output_fifo_re,
          north_output_fifo_ack,
215          east_output_fifo_rh,   east_output_fifo_ri,  east_output_fifo_re,
                east_output_fifo_ack,
216          south_output_fifo_rh, south_output_fifo_ri, south_output_fifo_re,
                south_output_fifo_ack,
217          west_output_fifo_rh,   west_output_fifo_ri,  west_output_fifo_re,
                west_output_fifo_ack,
218          local_output_fifo_rh, local_output_fifo_ri, local_output_fifo_re,
                local_output_fifo_ack  : std_logic;
219
```

```
220     -- output fifo data signals
221     signal north_output_fifo_data, east_output_fifo_data,
            south_output_fifo_data,
222            west_output_fifo_data,   local_output_fifo_data
                          : flit_data;
223
224     attribute keep : string;
225       attribute keep of north_east_rh,  north_east_ri,  north_east_re,
              north_east_ack,
226                         north_south_rh, north_south_ri, north_south_re,
                              north_south_ack,
227                         north_west_rh,    north_west_ri,  north_west_re,
                              north_west_ack,
228                         north_local_rh, north_local_ri, north_local_re,
                              north_local_ack,
229                         east_south_rh,  east_south_ri,  east_south_re,
                              east_south_ack,
230                         east_west_rh,   east_west_ri,   east_west_re,
                              east_west_ack,
231                         east_north_rh,    east_north_ri,  east_north_re,
                              east_north_ack,
232                         east_local_rh,    east_local_ri,  east_local_re,
                              east_local_ack,
233                         south_west_rh,    south_west_ri,  south_west_re,
                              south_west_ack,
234                         south_north_rh, south_north_ri, south_north_re,
                              south_north_ack,
235                         south_east_rh,    south_east_ri,  south_east_re,
                              south_east_ack,
236                         south_local_rh, south_local_ri, south_local_re,
                              south_local_ack,
237                         west_north_rh,    west_north_ri,  west_north_re,
                              west_north_ack,
238                         west_east_rh,   west_east_ri,   west_east_re,
                              west_east_ack,
239                         west_south_rh,    west_south_ri,  west_south_re,
                              west_south_ack,
240                         west_local_rh,    west_local_ri,  west_local_re,
                              west_local_ack,
241                         local_north_rh, local_north_ri, local_north_re,
                              local_north_ack,
242                         local_east_rh,    local_east_ri,  local_east_re,
                              local_east_ack,
243                         local_south_rh, local_south_ri, local_south_re,
                              local_south_ack,
244                         local_west_rh,    local_west_ri,  local_west_re,
                              local_west_ack  : signal is "true";
245
246     attribute keep of   north_east_data,  north_south_data, north_west_data,
            north_local_data,
247                         east_south_data,  east_west_data,   east_north_data,
                              east_local_data,
248                         south_west_data,  south_north_data, south_east_data,
                              south_local_data,
249                         west_north_data,  west_east_data,   west_south_data,
                              west_local_data,
250                         local_north_data, local_east_data,  local_south_data,
                              local_west_data : signal is "true";
251
252     attribute keep of   north_input_fifo_rh, north_input_fifo_ri,
            north_input_fifo_re, north_input_fifo_ack,
253                         east_input_fifo_rh,   east_input_fifo_ri,
                              east_input_fifo_re,  east_input_fifo_ack,
254                         south_input_fifo_rh, south_input_fifo_ri,
                              south_input_fifo_re, south_input_fifo_ack,
```

```
255                       west_input_fifo_rh,   west_input_fifo_ri,
                              west_input_fifo_re,  west_input_fifo_ack,
256                       local_input_fifo_rh, local_input_fifo_ri,
                              local_input_fifo_re, local_input_fifo_ack : signal
                               is "true";
257
258     attribute keep of   north_input_fifo_data, east_input_fifo_data,
            south_input_fifo_data,
259                       west_input_fifo_data,   local_input_fifo_data : signal
                              is "true";
260
261     attribute keep of   north_output_fifo_rh, north_output_fifo_ri,
            north_output_fifo_re, north_output_fifo_ack,
262                       east_output_fifo_rh,  east_output_fifo_ri,
                              east_output_fifo_re,  east_output_fifo_ack,
263                       south_output_fifo_rh, south_output_fifo_ri,
                              south_output_fifo_re, south_output_fifo_ack,
264                       west_output_fifo_rh,  west_output_fifo_ri,
                              west_output_fifo_re,  west_output_fifo_ack,
265                       local_output_fifo_rh, local_output_fifo_ri,
                              local_output_fifo_re, local_output_fifo_ack :
                              signal is "true";
266
267     attribute keep of   north_output_fifo_data, east_output_fifo_data,
            south_output_fifo_data,
268                       west_output_fifo_data,  local_output_fifo_data : signal
                               is "true";
269
270   begin
271
272     -- Input fifos --
273       north_input_fifo : fifo
274       generic map(
275         depth => 1
276       )
277       port map(
278         reset     => reset,
279         rh_in     => north_rh_in,
280         ri_in     => north_ri_in,
281         re_in     => north_re_in,
282         ack_in    => north_ack_in,
283         data_in   => north_data_in,
284         rh_out    => north_input_fifo_rh,
285         ri_out    => north_input_fifo_ri,
286         re_out    => north_input_fifo_re,
287         ack_out   => north_input_fifo_ack,
288         data_out  => north_input_fifo_data
289       );
290
291     east_input_fifo : fifo
292     generic map(
293       depth => 1
294     )
295     port map(
296       reset     => reset,
297       rh_in     => east_rh_in,
298       ri_in     => east_ri_in,
299       re_in     => east_re_in,
300       ack_in    => east_ack_in,
301       data_in   => east_data_in,
302       rh_out    => east_input_fifo_rh,
303       ri_out    => east_input_fifo_ri,
304       re_out    => east_input_fifo_re,
305       ack_out   => east_input_fifo_ack,
306       data_out  => east_input_fifo_data
```

```
307   );
308
309   south_input_fifo : fifo
310   generic map(
311     depth => 1
312   )
313   port map(
314     reset     => reset,
315     rh_in     => south_rh_in,
316     ri_in     => south_ri_in,
317     re_in     => south_re_in,
318     ack_in    => south_ack_in,
319     data_in   => south_data_in,
320     rh_out    => south_input_fifo_rh,
321     ri_out    => south_input_fifo_ri,
322     re_out    => south_input_fifo_re,
323     ack_out   => south_input_fifo_ack,
324     data_out  => south_input_fifo_data
325   );
326
327   west_input_fifo : fifo
328   generic map(
329     depth => 1
330   )
331   port map(
332     reset     => reset,
333     rh_in     => west_rh_in,
334     ri_in     => west_ri_in,
335     re_in     => west_re_in,
336     ack_in    => west_ack_in,
337     data_in   => west_data_in,
338     rh_out    => west_input_fifo_rh,
339     ri_out    => west_input_fifo_ri,
340     re_out    => west_input_fifo_re,
341     ack_out   => west_input_fifo_ack,
342     data_out  => west_input_fifo_data
343   );
344
345   local_input_fifo : fifo
346   generic map(
347     depth => 1
348   )
349   port map(
350     reset     => reset,
351     rh_in     => local_rh_in,
352     ri_in     => local_ri_in,
353     re_in     => local_re_in,
354     ack_in    => local_ack_in,
355     data_in   => local_data_in,
356     rh_out    => local_input_fifo_rh,
357     ri_out    => local_input_fifo_ri,
358     re_out    => local_input_fifo_re,
359     ack_out   => local_input_fifo_ack,
360     data_out  => local_input_fifo_data
361   );
362
363   -- Output fifos --
364     north_output_fifo : fifo
365     generic map(
366       depth => 1
367     )
368     port map(
369       reset     => reset,
370       rh_in     => north_output_fifo_rh,
371       ri_in     => north_output_fifo_ri,
```

```
372         re_in     => north_output_fifo_re,
373         ack_in    => north_output_fifo_ack,
374         data_in   => north_output_fifo_data,
375         rh_out    => north_rh_out,
376         ri_out    => north_ri_out,
377         re_out    => north_re_out,
378         ack_out   => north_ack_out,
379         data_out  => north_data_out
380       );
381
382     east_output_fifo : fifo
383     generic map(
384       depth => 1
385     )
386     port map(
387       reset     => reset,
388       rh_in     => east_output_fifo_rh,
389       ri_in     => east_output_fifo_ri,
390       re_in     => east_output_fifo_re,
391       ack_in    => east_output_fifo_ack,
392       data_in   => east_output_fifo_data,
393       rh_out    => east_rh_out,
394       ri_out    => east_ri_out,
395       re_out    => east_re_out,
396       ack_out   => east_ack_out,
397       data_out  => east_data_out
398     );
399
400     south_output_fifo : fifo
401     generic map(
402       depth => 1
403     )
404     port map(
405       reset     => reset,
406       rh_in     => south_output_fifo_rh,
407       ri_in     => south_output_fifo_ri,
408       re_in     => south_output_fifo_re,
409       ack_in    => south_output_fifo_ack,
410       data_in   => south_output_fifo_data,
411       rh_out    => south_rh_out,
412       ri_out    => south_ri_out,
413       re_out    => south_re_out,
414       ack_out   => south_ack_out,
415       data_out  => south_data_out
416     );
417
418     west_output_fifo : fifo
419     generic map(
420       depth => 1
421     )
422     port map(
423       reset     => reset,
424       rh_in     => west_output_fifo_rh,
425       ri_in     => west_output_fifo_ri,
426       re_in     => west_output_fifo_re,
427       ack_in    => west_output_fifo_ack,
428       data_in   => west_output_fifo_data,
429       rh_out    => west_rh_out,
430       ri_out    => west_ri_out,
431       re_out    => west_re_out,
432       ack_out   => west_ack_out,
433       data_out  => west_data_out
434     );
435
436     local_output_fifo : fifo
```

```
437   generic map(
438     depth => 1
439   )
440   port map(
441     reset     => reset,
442     rh_in     => local_output_fifo_rh,
443     ri_in     => local_output_fifo_ri,
444     re_in     => local_output_fifo_re,
445     ack_in    => local_output_fifo_ack,
446     data_in   => local_output_fifo_data,
447     rh_out    => local_rh_out,
448     ri_out    => local_ri_out,
449     re_out    => local_re_out,
450     ack_out   => local_ack_out,
451     data_out  => local_data_out
452   );
453
454   -- Input port instanciations --
455
456     north_input_port : input_port
457     port map (
458       reset       => reset,
459       -- north input channel --
460       rh_in       => north_input_fifo_rh,
461       ri_in       => north_input_fifo_ri,
462       re_in       => north_input_fifo_re,
463       ack_in      => north_input_fifo_ack,
464       data_in     => north_input_fifo_data,
465       -- local channel --
466       a_rh_out    => north_local_rh,
467       a_ri_out    => north_local_ri,
468       a_re_out    => north_local_re,
469       a_ack_out   => north_local_ack,
470       a_data_out  => north_local_data,
471       -- east channel --
472       b_rh_out    => north_east_rh,
473       b_ri_out    => north_east_ri,
474       b_re_out    => north_east_re,
475       b_ack_out   => north_east_ack,
476       b_data_out  => north_east_data,
477       -- south channel --
478       c_rh_out    => north_south_rh,
479       c_ri_out    => north_south_ri,
480       c_re_out    => north_south_re,
481       c_ack_out   => north_south_ack,
482       c_data_out  => north_south_data,
483       -- west channel --
484       d_rh_out    => north_west_rh,
485       d_ri_out    => north_west_ri,
486       d_re_out    => north_west_re,
487       d_ack_out   => north_west_ack,
488       d_data_out  => north_west_data
489     );
490
491   east_input_port : input_port
492   port map (
493     reset       => reset,
494     -- east input channel --
495     rh_in       => east_input_fifo_rh,
496     ri_in       => east_input_fifo_ri,
497     re_in       => east_input_fifo_re,
498     ack_in      => east_input_fifo_ack,
499     data_in     => east_input_fifo_data,
500     -- north channel --
501     a_rh_out    => east_north_rh,
```

```
502       a_ri_out    => east_north_ri,
503       a_re_out    => east_north_re,
504       a_ack_out   => east_north_ack,
505       a_data_out  => east_north_data,
506       -- local channel --
507       b_rh_out    => east_local_rh,
508       b_ri_out    => east_local_ri,
509       b_re_out    => east_local_re,
510       b_ack_out   => east_local_ack,
511       b_data_out  => east_local_data,
512       -- south channel --
513       c_rh_out    => east_south_rh,
514       c_ri_out    => east_south_ri,
515       c_re_out    => east_south_re,
516       c_ack_out   => east_south_ack,
517       c_data_out  => east_south_data,
518       -- west channel --
519       d_rh_out    => east_west_rh,
520       d_ri_out    => east_west_ri,
521       d_re_out    => east_west_re,
522       d_ack_out   => east_west_ack,
523       d_data_out  => east_west_data
524     );
525
526     south_input_port : input_port
527     port map (
528       reset       => reset,
529       -- south input channel --
530       rh_in       => south_input_fifo_rh,
531       ri_in       => south_input_fifo_ri,
532       re_in       => south_input_fifo_re,
533       ack_in      => south_input_fifo_ack,
534       data_in     => south_input_fifo_data,
535       -- north channel --
536       a_rh_out    => south_north_rh,
537       a_ri_out    => south_north_ri,
538       a_re_out    => south_north_re,
539       a_ack_out   => south_north_ack,
540       a_data_out  => south_north_data,
541       -- east channel --
542       b_rh_out    => south_east_rh,
543       b_ri_out    => south_east_ri,
544       b_re_out    => south_east_re,
545       b_ack_out   => south_east_ack,
546       b_data_out  => south_east_data,
547       -- local channel --
548       c_rh_out    => south_local_rh,
549       c_ri_out    => south_local_ri,
550       c_re_out    => south_local_re,
551       c_ack_out   => south_local_ack,
552       c_data_out  => south_local_data,
553       -- west channel --
554       d_rh_out    => south_west_rh,
555       d_ri_out    => south_west_ri,
556       d_re_out    => south_west_re,
557       d_ack_out   => south_west_ack,
558       d_data_out  => south_west_data
559     );
560
561     west_input_port : input_port
562     port map (
563       reset       => reset,
564       -- west input channel --
565       rh_in       => west_input_fifo_rh,
566       ri_in       => west_input_fifo_ri,
```

```
567       re_in       => west_input_fifo_re,
568       ack_in      => west_input_fifo_ack,
569       data_in     => west_input_fifo_data,
570       -- north channel --
571       a_rh_out    => west_north_rh,
572       a_ri_out    => west_north_ri,
573       a_re_out    => west_north_re,
574       a_ack_out   => west_north_ack,
575       a_data_out  => west_north_data,
576       -- east channel --
577       b_rh_out    => west_east_rh,
578       b_ri_out    => west_east_ri,
579       b_re_out    => west_east_re,
580       b_ack_out   => west_east_ack,
581       b_data_out  => west_east_data,
582       -- south channel --
583       c_rh_out    => west_south_rh,
584       c_ri_out    => west_south_ri,
585       c_re_out    => west_south_re,
586       c_ack_out   => west_south_ack,
587       c_data_out  => west_south_data,
588       -- local channel --
589       d_rh_out    => west_local_rh,
590       d_ri_out    => west_local_ri,
591       d_re_out    => west_local_re,
592       d_ack_out   => west_local_ack,
593       d_data_out  => west_local_data
594     );
595
596     local_input_port : input_port
597     port map (
598       reset       => reset,
599       -- local input channel --
600       rh_in       => local_input_fifo_rh,
601       ri_in       => local_input_fifo_ri,
602       re_in       => local_input_fifo_re,
603       ack_in      => local_input_fifo_ack,
604       data_in     => local_input_fifo_data,
605       -- north channel --
606       a_rh_out    => local_north_rh,
607       a_ri_out    => local_north_ri,
608       a_re_out    => local_north_re,
609       a_ack_out   => local_north_ack,
610       a_data_out  => local_north_data,
611       -- east channel --
612       b_rh_out    => local_east_rh,
613       b_ri_out    => local_east_ri,
614       b_re_out    => local_east_re,
615       b_ack_out   => local_east_ack,
616       b_data_out  => local_east_data,
617       -- south channel --
618       c_rh_out    => local_south_rh,
619       c_ri_out    => local_south_ri,
620       c_re_out    => local_south_re,
621       c_ack_out   => local_south_ack,
622       c_data_out  => local_south_data,
623       -- west channel --
624       d_rh_out    => local_west_rh,
625       d_ri_out    => local_west_ri,
626       d_re_out    => local_west_re,
627       d_ack_out   => local_west_ack,
628       d_data_out  => local_west_data
629     );
630
631     -- Output port instanciations --
```

```
632
633       north_output_port : output_port
634       port map(
635         reset   => reset,
636
637         a_rh    => east_north_rh,
638         a_ri    => east_north_ri,
639         a_re    => east_north_re,
640         a_ack   => east_north_ack,
641         a_data  => east_north_data,
642
643         b_rh    => south_north_rh,
644         b_ri    => south_north_ri,
645         b_re    => south_north_re,
646         b_ack   => south_north_ack,
647         b_data  => south_north_data,
648
649         c_rh    => west_north_rh,
650         c_ri    => west_north_ri,
651         c_re    => west_north_re,
652         c_ack   => west_north_ack,
653         c_data  => west_north_data,
654
655         d_rh    => local_north_rh,
656         d_ri    => local_north_ri,
657         d_re    => local_north_re,
658         d_ack   => local_north_ack,
659         d_data  => local_north_data,
660
661         output_rh   => north_output_fifo_rh,
662         output_ri   => north_output_fifo_ri,
663         output_re   => north_output_fifo_re,
664         output_ack  => north_output_fifo_ack,
665         output_data => north_output_fifo_data
666       );
667
668     east_output_port : output_port
669     port map(
670       reset   => reset,
671
672       a_rh    => north_east_rh,
673       a_ri    => north_east_ri,
674       a_re    => north_east_re,
675       a_ack   => north_east_ack,
676       a_data  => north_east_data,
677
678       b_rh    => south_east_rh,
679       b_ri    => south_east_ri,
680       b_re    => south_east_re,
681       b_ack   => south_east_ack,
682       b_data  => south_east_data,
683
684       c_rh    => west_east_rh,
685       c_ri    => west_east_ri,
686       c_re    => west_east_re,
687       c_ack   => west_east_ack,
688       c_data  => west_east_data,
689
690       d_rh    => local_east_rh,
691       d_ri    => local_east_ri,
692       d_re    => local_east_re,
693       d_ack   => local_east_ack,
694       d_data  => local_east_data,
695
696       output_rh   => east_output_fifo_rh,
```

```
697       output_ri   => east_output_fifo_ri,
698       output_re   => east_output_fifo_re,
699       output_ack  => east_output_fifo_ack,
700       output_data => east_output_fifo_data
701     );
702
703     south_output_port : output_port
704     port map(
705       reset   => reset,
706
707       a_rh    => north_south_rh,
708       a_ri    => north_south_ri,
709       a_re    => north_south_re,
710       a_ack   => north_south_ack,
711       a_data  => north_south_data,
712
713       b_rh    => east_south_rh,
714       b_ri    => east_south_ri,
715       b_re    => east_south_re,
716       b_ack   => east_south_ack,
717       b_data  => east_south_data,
718
719       c_rh    => west_south_rh,
720       c_ri    => west_south_ri,
721       c_re    => west_south_re,
722       c_ack   => west_south_ack,
723       c_data  => west_south_data,
724
725       d_rh    => local_south_rh,
726       d_ri    => local_south_ri,
727       d_re    => local_south_re,
728       d_ack   => local_south_ack,
729       d_data  => local_south_data,
730
731       output_rh   => south_output_fifo_rh,
732       output_ri   => south_output_fifo_ri,
733       output_re   => south_output_fifo_re,
734       output_ack  => south_output_fifo_ack,
735       output_data => south_output_fifo_data
736     );
737
738     west_output_port : output_port
739     port map(
740       reset   => reset,
741
742       a_rh    => north_west_rh,
743       a_ri    => north_west_ri,
744       a_re    => north_west_re,
745       a_ack   => north_west_ack,
746       a_data  => north_west_data,
747
748       b_rh    => east_west_rh,
749       b_ri    => east_west_ri,
750       b_re    => east_west_re,
751       b_ack   => east_west_ack,
752       b_data  => east_west_data,
753
754       c_rh    => south_west_rh,
755       c_ri    => south_west_ri,
756       c_re    => south_west_re,
757       c_ack   => south_west_ack,
758       c_data  => south_west_data,
759
760       d_rh    => local_west_rh,
761       d_ri    => local_west_ri,
```

```
762       d_re    => local_west_re,
763       d_ack   => local_west_ack,
764       d_data  => local_west_data,
765
766       output_rh   => west_output_fifo_rh,
767       output_ri   => west_output_fifo_ri,
768       output_re   => west_output_fifo_re,
769       output_ack  => west_output_fifo_ack,
770       output_data => west_output_fifo_data
771     );
772
773     local_output_port : output_port
774     port map(
775       reset   => reset,
776
777       a_rh    => north_local_rh,
778       a_ri    => north_local_ri,
779       a_re    => north_local_re,
780       a_ack   => north_local_ack,
781       a_data  => north_local_data,
782
783       b_rh    => east_local_rh,
784       b_ri    => east_local_ri,
785       b_re    => east_local_re,
786       b_ack   => east_local_ack,
787       b_data  => east_local_data,
788
789       c_rh    => south_local_rh,
790       c_ri    => south_local_ri,
791       c_re    => south_local_re,
792       c_ack   => south_local_ack,
793       c_data  => south_local_data,
794
795       d_rh    => west_local_rh,
796       d_ri    => west_local_ri,
797       d_re    => west_local_re,
798       d_ack   => west_local_ack,
799       d_data  => west_local_data,
800
801       output_rh   => local_output_fifo_rh,
802       output_ri   => local_output_fifo_ri,
803       output_re   => local_output_fifo_re,
804       output_ack  => local_output_fifo_ack,
805       output_data => local_output_fifo_data
806     );
807
808   end struct;
```

A.5.2.2 fifo.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   entity fifo is
8     generic(
9       depth : positive := 12
10    );
11    port(
12      reset  : in std_logic;
13      rh_in : in  STD_LOGIC;
```

```
14      ri_in : in  STD_LOGIC;
15      re_in : in  STD_LOGIC;
16      ack_in : out  STD_LOGIC;
17      data_in : in  flit_data;
18      rh_out : out  STD_LOGIC;
19      ri_out : out  STD_LOGIC;
20      re_out : out  STD_LOGIC;
21      ack_out : in  STD_LOGIC;
22      data_out : out  flit_data
23    );
24  end fifo;
25
26  architecture struct2 of fifo is
27
28    component fifo_stage is
29      Port (
30        reset : in  std_logic;
31        rh_in : in  STD_LOGIC;
32        ri_in : in  STD_LOGIC;
33        re_in : in  STD_LOGIC;
34        ack_in : out  STD_LOGIC;
35        data_in : in  flit_data;
36        rh_out : out  STD_LOGIC;
37        ri_out : out  STD_LOGIC;
38        re_out : out  STD_LOGIC;
39        ack_out : in  STD_LOGIC;
40        data_out : out  flit_data
41      );
42    end component;
43
44    type data_array is array (0 to depth) of flit_data;
45    signal data : data_array;
46
47    signal rh, ri, re, ack : std_logic_vector(depth downto 0);
48
49    attribute keep : string;
50    attribute keep of data : signal is "true";
51    attribute keep of rh   : signal is "true";
52    attribute keep of ri   : signal is "true";
53    attribute keep of re   : signal is "true";
54    attribute keep of ack  : signal is "true";
55
56  begin
57
58    --generate fifo chain
59    fifo_chain : for index in 0 to depth-1 generate
60
61    begin
62
63      stage : fifo_stage
64      port map(
65        reset     => reset,
66        rh_in     => rh(index),
67        ri_in     => ri(index),
68        re_in     => re(index),
69        ack_in    => ack(index),
70        data_in   => data(index),
71        rh_out    => rh(index+1),
72        ri_out    => ri(index+1),
73        re_out    => re(index+1),
74        ack_out   => ack(index+1),
75        data_out  => data(index+1)
76      );
77
78    end generate fifo_chain;
```

```
79
80
81      --Assign inputs
82    rh(0) <= rh_in;
83    ri(0) <= ri_in;
84    re(0) <= re_in;
85    ack_in  <= ack(0);
86    data(0) <= data_in;
87
88    --Assign outputs
89    rh_out      <= rh(depth);
90    ri_out      <= ri(depth);
91    re_out      <= re(depth);
92    ack(depth)  <= ack_out;
93    data_out    <= data(depth);
94
95  end struct2;
```

A.5.2.3 fifo_stage.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   library UNISIM;
8   use UNISIM.VComponents.lut4_l;
9
10  entity fifo_stage is
11    port(
12      reset     : in  std_logic;
13      rh_in     : in  STD_LOGIC;
14      ri_in     : in  STD_LOGIC;
15      re_in     : in  STD_LOGIC;
16      ack_in    : out STD_LOGIC;
17      data_in   : in  flit_data;
18      rh_out    : out STD_LOGIC;
19      ri_out    : out STD_LOGIC;
20      re_out    : out STD_LOGIC;
21      ack_out   : in  STD_LOGIC;
22      data_out  : out flit_data
23    );
24  end fifo_stage;
25
26  architecture Behavioral of fifo_stage is
27
28    component as_bd_4p_delay is
29      generic(
30        size : natural := 10  -- Delay size
31      );
32      port (
33        d : in  std_logic;    -- Data in
34        z : out std_logic     -- Data out
35      );
36    end component;
37
38    component as_bd_4p_c2 is
39      port (
40        reset : in  std_logic;    -- Reset (Active low)
41        a     : in  std_logic;    -- Input A
42        b     : in  std_logic;    -- Input B
43        z     : out std_logic     -- Output Z
```

```
44      );
45    end component;
46
47    signal rh_int,ri_int,re_int, rh_int_delayed, ri_int_delayed, re_int_delayed
          , latch_enable, ack_in_int, not_ack : std_logic;
48
49    attribute keep : string;
50    attribute keep of rh_int          : signal is "true";
51    attribute keep of ri_int          : signal is "true";
52    attribute keep of re_int          : signal is "true";
53    attribute keep of rh_int_delayed  : signal is "true";
54    attribute keep of ri_int_delayed  : signal is "true";
55    attribute keep of re_int_delayed  : signal is "true";
56    attribute keep of latch_enable    : signal is "true";
57    attribute keep of ack_in_int      : signal is "true";
58
59
60
61  begin
62
63    latch_enable <= rh_int or ri_int or re_int or ack_out;
64    ack_in <= rh_int or ri_int or re_int;
65    not_ack <= not ack_out;
66
67    rh_c : as_bd_4p_c2
68    port map(
69      reset => reset,
70      a     => rh_in,
71      b     => not_ack,
72      z     => rh_int
73    );
74
75    ri_c : as_bd_4p_c2
76    port map(
77      reset => reset,
78      a     => ri_in,
79      b     => not_ack,
80      z     => ri_int
81    );
82
83    re_c : as_bd_4p_c2
84    port map(
85      reset => reset,
86      a     => re_in,
87      b     => not_ack,
88      z     => re_int
89    );
90
91    -- delay element
92    rh_delay : as_bd_4p_delay
93    generic map(
94      size => 5
95    )
96    port map(
97      d => rh_int,
98      z => rh_int_delayed
99    );
100
101   -- delay element
102   ri_delay : as_bd_4p_delay
103   generic map(
104     size => 5
105   )
106   port map(
107     d => ri_int,
```

```
108       z => ri_int_delayed
109     );
110
111     -- delay element
112     re_delay : as_bd_4p_delay
113     generic map(
114       size => 5
115     )
116     port map(
117       d => re_int,
118       z => re_int_delayed
119     );
120
121     rh_out <= rh_int_delayed;
122     ri_out <= ri_int_delayed;
123     re_out <= re_int_delayed;
124
125     data_latch : process(latch_enable, data_in)
126     begin
127       if latch_enable = '0' then
128         data_out <= data_in;
129       end if;
130     end process;
131
132   end Behavioral;
```

A.5.2.4 input_port.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   library UNISIM;
8   use UNISIM.VComponents.lut2;
9   use UNISIM.VComponents.lut4_l;
10
11  entity input_port is
12    port (
13      reset       : in std_logic;
14      -- Input channel --
15      rh_in       : in  std_logic;
16      ri_in       : in  std_logic;
17      re_in       : in  std_logic;
18      ack_in      : out std_logic;
19      data_in     : in  flit_data;
20      -- Output channel A --
21      a_rh_out    : out std_logic;
22      a_ri_out    : out std_logic;
23      a_re_out    : out std_logic;
24      a_ack_out   : in  std_logic;
25      a_data_out  : out flit_data;
26      -- Output channel B --
27      b_rh_out    : out std_logic;
28      b_ri_out    : out std_logic;
29      b_re_out    : out std_logic;
30      b_ack_out   : in  std_logic;
31      b_data_out  : out flit_data;
32      -- Output channel C --
33      c_rh_out    : out std_logic;
34      c_ri_out    : out std_logic;
35      c_re_out    : out std_logic;
```

```
36      c_ack_out   : in  std_logic;
37      c_data_out  : out flit_data;
38      -- Output channel LOCAL --
39      d_rh_out    : out std_logic;
40      d_ri_out    : out std_logic;
41      d_re_out    : out std_logic;
42      d_ack_out   : in  std_logic;
43      d_data_out  : out flit_data
44    );
45  end input_port;
46
47
48
49  architecture structural of input_port is
50
51    component as_bd_4p_delay is
52      generic(
53        size : natural := 10  -- Delay size
54      );
55      port (
56        d : in  std_logic;    -- Data in
57        z : out std_logic     -- Data out
58      );
59    end component;
60
61    component as_bd_4p_c2 is
62      port (
63        reset : in  std_logic;    -- Reset (Active low)
64        a     : in  std_logic;    -- Input A
65        b     : in  std_logic;    -- Input B
66        z     : out std_logic     -- Output Z
67      );
68    end component;
69
70    component header_rotater is
71      port(
72        reset      : in  std_logic;
73        req_header : in  std_logic;
74        ack        : in  std_logic;
75        data_in    : in  flit_data;
76        data_out   : out flit_data
77      );
78    end component;
79
80    -- Internal signals --
81    signal data_out : flit_data;
82    signal rh_in_delayed, ri_in_delayed, re_in_delayed : std_logic;
83    signal ack : std_logic;
84    signal route_addr : std_logic_vector(1 downto 0);
85    signal head_rotate : std_logic;
86
87    attribute keep : string;
88    attribute keep of data_out      : signal is "true";
89    attribute keep of rh_in_delayed : signal is "true";
90    attribute keep of ri_in_delayed : signal is "true";
91    attribute keep of re_in_delayed : signal is "true";
92    attribute keep of ack           : signal is "true";
93    attribute keep of route_addr    : signal is "true";
94    attribute keep of head_rotate   : signal is "true";
95
96  begin
97
98  -- Header rotater --
99
100   h_rotater : header_rotater
```

```
101   port map(
102     reset      => reset,
103     req_header => rh_in,
104     ack        => ack,
105     data_in    => data_in,
106     data_out   => data_out
107   );
108
109 -- Latch routing address when rh_in is asserted
110 -- The routing address is the two MSBs of the header flit.
111   routing_addr_latch : process(rh_in, data_in)
112   begin
113     if rh_in = '1' then
114       route_addr <= data_in(FLIT_SIZE-1 downto FLIT_SIZE-2);
115     end if;
116   end process;
117
118 -- delay rh_in
119   rh_delay : as_bd_4p_delay
120   generic map(
121     size => 12
122   )
123   port map(
124     d => rh_in,
125     z => rh_in_delayed
126   );
127
128 -- delay ri_in
129   ri_delay : as_bd_4p_delay
130   generic map(
131     size => 5
132   )
133   port map(
134     d => ri_in,
135     z => ri_in_delayed
136   );
137
138 -- delay re_in
139   re_delay : as_bd_4p_delay
140   generic map(
141     size => 5
142   )
143   port map(
144     d => re_in,
145     z => re_in_delayed
146   );
147
148 -- acknowledge signal
149 ack_in <= ack;
150
151 -- DEMUX the input port to the four different output ports based on
        route_addr --
152 demux : process(route_addr, rh_in_delayed, ri_in_delayed, re_in_delayed,
        a_ack_out, b_ack_out, c_ack_out, d_ack_out)
153 begin
154   case route_addr is
155     when "00" =>      -- port a
156       a_rh_out  <= rh_in_delayed;
157       a_ri_out  <= ri_in_delayed;
158       a_re_out  <= re_in_delayed;
159       ack       <= a_ack_out;
160       b_rh_out  <= '0';
161       b_ri_out  <= '0';
162       b_re_out  <= '0';
163       c_rh_out  <= '0';
```

```
164         c_ri_out  <= '0';
165         c_re_out  <= '0';
166         d_rh_out  <= '0';
167         d_ri_out  <= '0';
168         d_re_out  <= '0';
169       when "01" =>        --port b
170         b_rh_out  <= rh_in_delayed;
171         b_ri_out  <= ri_in_delayed;
172         b_re_out  <= re_in_delayed;
173         ack       <= b_ack_out;
174         a_rh_out  <= '0';
175         a_ri_out  <= '0';
176         a_re_out  <= '0';
177         c_rh_out  <= '0';
178         c_ri_out  <= '0';
179         c_re_out  <= '0';
180         d_rh_out  <= '0';
181         d_ri_out  <= '0';
182         d_re_out  <= '0';
183       when "10" =>        --port c
184         c_rh_out  <= rh_in_delayed;
185         c_ri_out  <= ri_in_delayed;
186         c_re_out  <= re_in_delayed;
187         ack       <= c_ack_out;
188         a_rh_out  <= '0';
189         a_ri_out  <= '0';
190         a_re_out  <= '0';
191         b_rh_out  <= '0';
192         b_ri_out  <= '0';
193         b_re_out  <= '0';
194         d_rh_out  <= '0';
195         d_ri_out  <= '0';
196         d_re_out  <= '0';
197       when others =>     --port d
198         d_rh_out  <= rh_in_delayed;
199         d_ri_out  <= ri_in_delayed;
200         d_re_out  <= re_in_delayed;
201         ack       <= d_ack_out;
202         a_rh_out  <= '0';
203         a_ri_out  <= '0';
204         a_re_out  <= '0';
205         b_rh_out  <= '0';
206         b_ri_out  <= '0';
207         b_re_out  <= '0';
208         c_rh_out  <= '0';
209         c_ri_out  <= '0';
210         c_re_out  <= '0';
211     end case;
212   end process;
213
214
215   -- assign data_out to all output channels
216   a_data_out  <= data_out;
217   b_data_out  <= data_out;
218   c_data_out  <= data_out;
219   d_data_out  <= data_out;
220
221   end structural;
```

A.5.2.5 header_rotater.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
```

```
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  library UNISIM;
8  use UNISIM.VComponents.lut2;
9  use UNISIM.VComponents.lut4_l;
10
11 entity header_rotater is
12   port(
13     reset      : in  std_logic;
14     req_header : in  std_logic;
15     ack        : in  std_logic;
16     data_in    : in  flit_data;
17     data_out   : out flit_data
18   );
19 end header_rotater;
20
21 architecture struct of header_rotater is
22
23   component as_bd_4p_c2 is
24     port (
25       reset : in  std_logic;     -- Reset (Active low)
26       a     : in  std_logic;     -- Input A
27       b     : in  std_logic;     -- Input B
28       z     : out std_logic      -- Output Z
29     );
30   end component;
31
32   signal sel,c_out : std_logic;
33
34   attribute keep : string;
35   attribute keep of sel,c_out : signal is "true";
36
37 begin
38
39   -- Mux Control --
40
41   head_rotate_c : as_bd_4p_c2
42   port map(
43     reset => reset,
44     a => req_header,
45     b => ack,
46     z => c_out
47   );
48
49   sel <= c_out or req_header;
50
51   -- Move the two MSBs so they become LSBs if sel is asserted
52   with sel select
53     data_out <= data_in(FLIT_SIZE-3 downto 0) & data_in(FLIT_SIZE-1 downto
         FLIT_SIZE-2) when '1',
54                 data_in
                                 when others;
55
56 end struct;
```

A.5.2.6 output_port.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
5  use work.types.all;
6
7  entity output_port is
8    port(
9        reset   : in  std_logic;  --Reset (active low)
10
11       a_rh    : in  std_logic;  --Handshake signals for input a
12       a_ri    : in  std_logic;
13       a_re    : in  std_logic;
14       a_ack   : out std_logic;
15
16       b_rh    : in  std_logic;  --Handshake signals for input b
17       b_ri    : in  std_logic;
18       b_re    : in  std_logic;
19       b_ack   : out std_logic;
20
21       c_rh    : in  std_logic;  --Handshake signals for input c
22       c_ri    : in  std_logic;
23       c_re    : in  std_logic;
24       c_ack   : out std_logic;
25
26       d_rh    : in  std_logic;  --Handshake signals for IP input
27       d_ri    : in  std_logic;
28       d_re    : in  std_logic;
29       d_ack   : out std_logic;
30
31       output_rh   : out std_logic;  --Handshake signals for output port
32       output_ri   : out std_logic;
33       output_re   : out std_logic;
34       output_ack  : in  std_logic;
35
36       a_data      : in  flit_data;  -- Data signals
37       b_data      : in  flit_data;
38       c_data      : in  flit_data;
39       d_data      : in  flit_data;
40       output_data : out flit_data
41     );
42 end output_port;
43
44 architecture Behavioral of output_port is
45
46   component as_bd_4p_delay is
47     generic(
48       size : natural := 10  -- Delay size
49     );
50     port (
51       d : in  std_logic;    -- Data in
52       z : out std_logic     -- Data out
53     );
54   end component;
55
56   component access_control is
57     port(
58       reset   : in  std_logic;  -- Reset (Active low)
59       rh_in   : in  std_logic;  -- header request input
60       ri_in   : in  std_logic;  -- intermediate request input
61       re_in   : in  std_logic;  -- end of packet request input
62       ack_in  : out std_logic;  -- ack input
63       rh_out : out  std_logic;  -- header request output
64       ri_out : out std_logic;   -- intermediate request output
65       re_out : out std_logic;   -- end of packet request output
66       ack_out : in  std_logic;  -- ack output
67       m_req   : out std_logic;  -- Reqeust access
68       m_grant : in  std_logic   -- Access granted
69     );
```

```
70    end component;
71
72    component merge4 is
73      port(
74        reset   : in  std_logic;  --Reset (active low)
75
76        a_rh    : in  std_logic;  --Handshake signals for input a
77        a_ri    : in  std_logic;
78        a_re    : in  std_logic;
79        a_ack   : out std_logic;
80
81        b_rh    : in  std_logic;  --Handshake signals for input b
82        b_ri    : in  std_logic;
83        b_re    : in  std_logic;
84        b_ack   : out std_logic;
85
86        c_rh    : in  std_logic;  --Handshake signals for input c
87        c_ri    : in  std_logic;
88        c_re    : in  std_logic;
89        c_ack   : out std_logic;
90
91        d_rh    : in  std_logic;  --Handshake signals for input d
92        d_ri    : in  std_logic;
93        d_re    : in  std_logic;
94        d_ack   : out std_logic;
95
96        z_rh    : out std_logic;  --Handshake signals for output z
97        z_ri    : out std_logic;
98        z_re    : out std_logic;
99        z_ack   : in  std_logic;
100
101       a_data  : in  flit_data;  -- Data signals
102       b_data  : in  flit_data;
103       c_data  : in  flit_data;
104       d_data  : in  flit_data;
105       z_data  : out flit_data
106     );
107   end component;
108
109   component mutex4 is
110     port (
111       reset : in  std_logic;
112       r1    : in  std_logic;
113       r2    : in  std_logic;
114       r3    : in  std_logic;
115       r4    : in  std_logic;
116       g1    : out std_logic;
117       g2    : out std_logic;
118       g3    : out std_logic;
119       g4    : out std_logic
120     );
121   end component;
122
123   -- Internal signals
124   signal  a_rh_int,   a_ri_int,   a_re_int,   a_ack_int,
125           b_rh_int,   b_ri_int,   b_re_int,   b_ack_int,
126           c_rh_int,   c_ri_int,   c_re_int,   c_ack_int,
127           d_rh_int,   d_ri_int,   d_re_int,   d_ack_int,
128           z_rh_int,   z_ri_int,   z_re_int              : std_logic;
129
130   signal a_m_req, b_m_req, c_m_req, d_m_req,
131          a_m_grant, b_m_grant, c_m_grant, d_m_grant : std_logic;
132
133
134   attribute keep : string;
```

```
135
136   attribute keep of   a_rh_int,   a_ri_int,   a_re_int,   a_ack_int,
137                       b_rh_int,   b_ri_int,   b_re_int,   b_ack_int,
138                       c_rh_int,   c_ri_int,   c_re_int,   c_ack_int,
139                       d_rh_int,   d_ri_int,   d_re_int,   d_ack_int,
140                       z_rh_int,   z_ri_int,   z_re_int,
141                       a_m_req,    b_m_req,    c_m_req,    d_m_req,
142                       a_m_grant,  b_m_grant,  c_m_grant,  d_m_grant : signal
                              is "true";
143
144 begin
145
146
147   a_access_control : access_control
148     port map (
149       reset   => reset,
150       rh_in   => a_rh,
151       ri_in   => a_ri,
152       re_in   => a_re,
153       ack_in  => a_ack,
154       rh_out => a_rh_int,
155       ri_out => a_ri_int,
156       re_out => a_re_int,
157       ack_out => a_ack_int,
158       m_req   => a_m_req,
159       m_grant => a_m_grant
160     );
161
162   b_access_control : access_control
163     port map (
164       reset   => reset,
165       rh_in   => b_rh,
166       ri_in   => b_ri,
167       re_in   => b_re,
168       ack_in  => b_ack,
169       rh_out => b_rh_int,
170       ri_out => b_ri_int,
171       re_out => b_re_int,
172       ack_out => b_ack_int,
173       m_req   => b_m_req,
174       m_grant => b_m_grant
175     );
176
177   c_access_control : access_control
178     port map (
179       reset   => reset,
180       rh_in   => c_rh,
181       ri_in   => c_ri,
182       re_in   => c_re,
183       ack_in  => c_ack,
184       rh_out => c_rh_int,
185       ri_out => c_ri_int,
186       re_out => c_re_int,
187       ack_out => c_ack_int,
188       m_req   => c_m_req,
189       m_grant => c_m_grant
190     );
191
192   d_access_control : access_control
193     port map (
194       reset   => reset,
195       rh_in   => d_rh,
196       ri_in   => d_ri,
197       re_in   => d_re,
198       ack_in  => d_ack,
```

```
199         rh_out => d_rh_int,
200         ri_out => d_ri_int,
201         re_out => d_re_int,
202         ack_out => d_ack_int,
203         m_req   => d_m_req,
204         m_grant => d_m_grant
205       );
206
207     mutex : mutex4
208       port map(
209         reset => reset,
210         r1    => a_m_req,
211         r2    => b_m_req,
212         r3    => c_m_req,
213         r4    => d_m_req,
214         g1    => a_m_grant,
215         g2    => b_m_grant,
216         g3    => c_m_grant,
217         g4    => d_m_grant
218       );
219
220     merge : merge4
221       port map (
222         reset   => reset,
223
224         a_rh    => a_rh_int,
225         a_ri    => a_ri_int,
226         a_re    => a_re_int,
227         a_ack   => a_ack_int,
228
229         b_rh    => b_rh_int,
230         b_ri    => b_ri_int,
231         b_re    => b_re_int,
232         b_ack   => b_ack_int,
233
234         c_rh    => c_rh_int,
235         c_ri    => c_ri_int,
236         c_re    => c_re_int,
237         c_ack   => c_ack_int,
238
239         d_rh    => d_rh_int,
240         d_ri    => d_ri_int,
241         d_re    => d_re_int,
242         d_ack   => d_ack_int,
243
244         z_rh    => z_rh_int,
245         z_ri    => z_ri_int,
246         z_re    => z_re_int,
247         z_ack   => output_ack,
248
249         a_data  => a_data,
250         b_data  => b_data,
251         c_data  => c_data,
252         d_data  => d_data,
253         z_data  => output_data
254       );
255
256     -- delay z_rh
257     z_rh_delay : as_bd_4p_delay
258     generic map(
259       size => 8
260     )
261     port map(
262       d => z_rh_int,
263       z => output_rh
```

```
264   );
265
266   -- delay z_ri
267   z_ri_delay : as_bd_4p_delay
268   generic map(
269     size => 8
270   )
271   port map(
272     d => z_ri_int,
273     z => output_ri
274   );
275
276   -- delay z_re
277   z_re_delay : as_bd_4p_delay
278   generic map(
279     size => 8
280   )
281   port map(
282     d => z_re_int,
283     z => output_re
284   );
285
286 end Behavioral;
```

A.5.2.7 access_control.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   library UNISIM;
7   use UNISIM.VComponents.lut2;
8   use UNISIM.VComponents.lut4_1;
9
10  entity access_control is
11    port(
12      reset   : in  std_logic;  -- Reset (Active low)
13      rh_in   : in  std_logic;  -- header request input
14      ri_in   : in  std_logic;  -- intermediate request input
15      re_in   : in  std_logic;  -- end of packet request input
16      ack_in  : out std_logic;  -- ack input
17      rh_out : out  std_logic;  -- header request output
18      ri_out : out std_logic;   -- intermediate request output
19      re_out : out std_logic; -- end of packet request output
20      ack_out : in  std_logic;  -- ack output
21      m_req   : out std_logic;  -- Reqeust access
22      m_grant : in  std_logic   -- Access granted
23    );
24  end access_control;
25
26  architecture Behavioral of access_control is
27
28    signal ack_in_internal, re_out_internal, ri_out_internal, csc0, four :
          std_logic;
29
30    attribute keep : string;
31    attribute keep of ack_in_internal, re_out_internal, ri_out_internal, csc0,
          four : signal is "true";
32
33    attribute rloc : string;
34    attribute rloc of rh_out_LUT  : label is "X0Y0";
35    attribute rloc of m_req_LUT   : label is "X0Y0";
```

```
36    attribute rloc of four_LUT    : label is "X0Y0";
37    attribute rloc of csc0_c      : label is "X0Y0";
38    attribute rloc of re_out_c    : label is "X1Y0";
39
40  begin
41
42  --# EQN file for model sky_grant_la
43  --# Generated by petrify 4.2 (compiled 15-Oct-03 at 3:06 PM)
44  --# Outputs between brackets "[out]" indicate a feedback to input "out"
45  --# Estimated area = 18.00
46  --
47  --INORDER = m_grant ack_out r_e_in r_i_in r_h_in ack_in r_e_out r_i_out
        r_h_out m_req csc0;
48  --OUTORDER = [ack_in] [r_e_out] [r_i_out] [r_h_out] [m_req] [csc0];
49  --[ack_in] = ack_out;
50  --[r_i_out] = r_i_in;
51  --[r_h_out] = r_h_in m_grant;
52  --[m_req] = csc0 + ack_in;
53  --[4] = r_e_out r_e_in';
54  --[csc0] = [4]' (r_h_in + csc0) + r_h_in csc0;       # mappable onto gC
55  --[r_e_out] = csc0 (r_e_in + r_e_out) + r_e_in r_e_out;      # mappable onto
        gC
56  --
57  --# Set/reset pins: reset(csc0)
58
59    ack_in_internal <= ack_out;
60    ri_out_internal <= ri_in;
61
62
63  --[r_h_out] = r_h_in m_grant;
64
65    rh_out_LUT: LUT2
66    generic map (
67      INIT => X"8")
68
69    port map (
70      O  => rh_out,
71      I0 => rh_in,
72      I1 => m_grant
73    );
74
75  --[m_req] = csc0 + ack_in;
76
77    m_req_LUT: LUT2
78    generic map (
79      INIT => X"e")
80
81    port map (
82      O  => m_req,
83      I0 => ack_in_internal,
84      I1 => csc0
85    );
86
87  --[4] = r_e_out r_e_in';
88
89    four_LUT: LUT2
90    generic map (
91      INIT => X"4")
92
93    port map (
94      O  => four,
95      I0 => re_in,
96      I1 => re_out_internal
97    );
98
```

```
99
100   -- C-element with inverted i1 input
101   csc0_c: lut4_l
102   generic map (
103     init => "10110010" & X"00"
104   )
105   port map (
106     i0 => rh_in,
107     i1 => four,
108     i2 => csc0,
109     i3 => reset,
110     lo => csc0
111   );
112
113   re_out_c: lut4_l
114   generic map (
115     init => "11101000" & X"00"
116   )
117   port map (
118     i0 => re_in,
119     i1 => csc0,
120     i2 => re_out_internal,
121     i3 => reset,
122     lo => re_out_internal
123   );
124
125   --Assign outputs
126   re_out <= re_out_internal;
127   ri_out <= ri_out_internal;
128   ack_in <= ack_in_internal;
129
130 end Behavioral;
```

A.5.2.8 mutex4.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity mutex4 is
7    port (
8      reset : in  std_logic;
9      r1    : in  std_logic;
10     r2    : in  std_logic;
11     r3    : in  std_logic;
12     r4    : in  std_logic;
13     g1    : out std_logic;
14     g2    : out std_logic;
15     g3    : out std_logic;
16     g4    : out std_logic
17   );
18
19 end mutex4;
20
21 architecture structural of mutex4 is
22
23   component as_bd_4p_mutex is
24     generic(
25       reset_value : bit := '0'; -- Reset value of output
26       --Only used for the random number generator for the behavioral arch.
27       --Must be seeded with some random value before simulating in modelsim
28       --by e.g. using a tcl script
```

```
29        seed1 : positive := 1;
30        seed2 : positive := 1
31      );
32      port (
33        reset : in  std_logic;     -- Reset (Active low)
34        r1,r2 : in  std_logic;     -- in
35        g1,g2 : out std_logic      -- mutex out
36      );
37    end component as_bd_4p_mutex;
38
39    -- Configure below which mutex architecture to use    --
40    -- Use as_fpga.as_bd_4p_mutex(behaviour) for simulation  --
41    -- Use as_fpga.as_bd_4p_mutex(gate) for synthesis       --
42    ------------------------------------------------------
43    --for all: as_bd_4p_mutex use entity work.as_bd_4p_mutex(gate);
44
45    signal r11, r12, r21, r22, r31, r32, r41, r42 : std_logic;
46
47    attribute keep : string;
48    attribute keep of r11 : signal is "true";
49    attribute keep of r12 : signal is "true";
50    attribute keep of r21 : signal is "true";
51    attribute keep of r22 : signal is "true";
52    attribute keep of r31 : signal is "true";
53    attribute keep of r32 : signal is "true";
54    attribute keep of r41 : signal is "true";
55    attribute keep of r42 : signal is "true";
56
57  begin
58
59    -- Implements a four-input mutexes from a net of 6 two-input mutexes.
60
61    mutex1 : as_bd_4p_mutex
62      port map(
63        reset => reset,
64        r1    => r1,
65        r2    => r2,
66        g1    => r11,
67        g2    => r21
68      );
69
70    mutex2 : as_bd_4p_mutex
71      port map(
72        reset => reset,
73        r1    => r3,
74        r2    => r4,
75        g1    => r31,
76        g2    => r41
77      );
78
79    mutex3 : as_bd_4p_mutex
80      port map(
81        reset => reset,
82        r1    => r11,
83        r2    => r31,
84        g1    => r12,
85        g2    => r32
86      );
87
88    mutex4 : as_bd_4p_mutex
89      port map(
90        reset => reset,
91        r1    => r21,
92        r2    => r41,
93        g1    => r22,
```

```
94        g2    => r42
95      );
96
97    mutex5 : as_bd_4p_mutex
98      port map(
99        reset => reset,
100       r1    => r22,
101       r2    => r32,
102       g1    => g2,
103       g2    => g3
104     );
105
106   mutex6 : as_bd_4p_mutex
107     port map(
108       reset => reset,
109       r1    => r12,
110       r2    => r42,
111       g1    => g1,
112       g2    => g4
113     );
114
115 end structural;
```

A.5.2.9 merge4.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7
8   entity merge4 is
9       port(
10        reset   : in  std_logic;  --Reset (active low)
11
12        a_rh    : in  std_logic;  --Handshake signals for input a
13        a_ri    : in  std_logic;
14        a_re    : in  std_logic;
15        a_ack   : out std_logic;
16
17        b_rh    : in  std_logic;  --Handshake signals for input b
18        b_ri    : in  std_logic;
19        b_re    : in  std_logic;
20        b_ack   : out std_logic;
21
22        c_rh    : in  std_logic;  --Handshake signals for input c
23        c_ri    : in  std_logic;
24        c_re    : in  std_logic;
25        c_ack   : out std_logic;
26
27        d_rh    : in  std_logic;  --Handshake signals for input d
28        d_ri    : in  std_logic;
29        d_re    : in  std_logic;
30        d_ack   : out std_logic;
31
32        z_rh    : out std_logic;  --Handshake signals for output z
33        z_ri    : out std_logic;
34        z_re    : out std_logic;
35        z_ack   : in  std_logic;
36
37        a_data  : in  flit_data;  -- Data signals
38        b_data  : in  flit_data;
```

```
39        c_data  : in  flit_data;
40        d_data  : in  flit_data;
41        z_data  : out flit_data
42      );
43  end merge4;
44
45  architecture structural of merge4 is
46
47    component as_bd_4p_c2 is
48      port (
49        reset : in  std_logic;     -- Reset (Active low)
50        a     : in  std_logic;     -- Input A
51        b     : in  std_logic;     -- Input B
52        z     : out std_logic      -- Output Z
53      );
54    end component;
55
56    -- Internal signals --
57    signal a_req, b_req, c_req, d_req, a_ack_int, b_ack_int, c_ack_int,
          d_ack_int : std_logic;
58    signal mux_control : std_logic_vector(3 downto 0);
59
60    attribute keep : string;
61    attribute keep of a_req      : signal is "true";
62    attribute keep of b_req      : signal is "true";
63    attribute keep of c_req      : signal is "true";
64    attribute keep of d_req      : signal is "true";
65    attribute keep of a_ack_int  : signal is "true";
66    attribute keep of b_ack_int  : signal is "true";
67    attribute keep of c_ack_int  : signal is "true";
68    attribute keep of d_ack_int  : signal is "true";
69    attribute keep of mux_control : signal is "true";
70
71  begin
72
73    z_rh <= a_rh or b_rh or c_rh or d_rh;
74    z_ri <= a_ri or b_ri or c_ri or d_ri;
75    z_re <= a_re or b_re or c_re or d_re;
76
77    a_req <= a_rh or a_ri or a_re;
78    b_req <= b_rh or b_ri or b_re;
79    c_req <= c_rh or c_ri or c_re;
80    d_req <= d_rh or d_ri or d_re;
81
82    a_ack <= a_ack_int;
83    b_ack <= b_ack_int;
84    c_ack <= c_ack_int;
85    d_ack <= d_ack_int;
86
87    a_ack_c_element : as_bd_4p_c2
88      port map (
89        reset => reset,
90        a     => z_ack,
91        b     => a_req,
92        z     => a_ack_int
93      );
94
95    b_ack_c_element : as_bd_4p_c2
96      port map (
97        reset => reset,
98        a     => z_ack,
99        b     => b_req,
100       z     => b_ack_int
101     );
102
```

```
103   c_ack_c_element : as_bd_4p_c2
104     port map (
105       reset => reset,
106       a     => z_ack,
107       b     => c_req,
108       z     => c_ack_int
109     );
110
111   d_ack_c_element : as_bd_4p_c2
112     port map (
113       reset => reset,
114       a     => z_ack,
115       b     => d_req,
116       z     => d_ack_int
117     );
118
119   data_mux : process(a_req, a_ack_int, b_req, b_ack_int, c_req, c_ack_int,
          d_req, d_ack_int, a_data, b_data, c_data, d_data)
120   begin
121     if    (a_req or a_ack_int) = '1' then
122       z_data <= a_data;
123     elsif (b_req or b_ack_int) = '1' then
124       z_data <= b_data;
125     elsif (c_req or c_ack_int) = '1' then
126       z_data <= c_data;
127     elsif (d_req or d_ack_int) = '1' then
128       z_data <= d_data;
129     else
130       z_data <= FLIT_ZERO;
131     end if;
132   end process;
133
134 end structural;
```

A.5.2.10 be_router_boardtest.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6   use work.source_rom_data.all;
7
8   entity be_router_boardtest is
9     port(
10      reset         : in  std_logic;
11      north_alive   : out std_logic;
12      east_alive    : out std_logic;
13      south_alive   : out std_logic;
14      west_alive    : out std_logic;
15      local_alive   : out std_logic
16    );
17  end be_router_boardtest;
18
19  architecture Behavioral of be_router_boardtest is
20
21    component be_router is
22      port(
23        reset : in std_logic;
24        -- Input ports --
25        north_rh_in   : in  std_logic;
26        north_ri_in   : in  std_logic;
27        north_re_in   : in  std_logic;
```

```
28        north_ack_in  : out std_logic;
29        north_data_in : in  flit_data;
30
31        west_rh_in    : in  std_logic;
32        west_ri_in    : in  std_logic;
33        west_re_in    : in  std_logic;
34        west_ack_in   : out std_logic;
35        west_data_in  : in  flit_data;
36
37        south_rh_in   : in  std_logic;
38        south_ri_in   : in  std_logic;
39        south_re_in   : in  std_logic;
40        south_ack_in  : out std_logic;
41        south_data_in : in  flit_data;
42
43        east_rh_in    : in  std_logic;
44        east_ri_in    : in  std_logic;
45        east_re_in    : in  std_logic;
46        east_ack_in   : out std_logic;
47        east_data_in  : in  flit_data;
48
49        local_rh_in   : in  std_logic;
50        local_ri_in   : in  std_logic;
51        local_re_in   : in  std_logic;
52        local_ack_in  : out std_logic;
53        local_data_in : in  flit_data;
54
55        -- Output ports --
56        north_rh_out    : out std_logic;
57        north_ri_out    : out std_logic;
58        north_re_out    : out std_logic;
59        north_ack_out   : in  std_logic;
60        north_data_out  : out flit_data;
61
62        west_rh_out     : out std_logic;
63        west_ri_out     : out std_logic;
64        west_re_out     : out std_logic;
65        west_ack_out    : in  std_logic;
66        west_data_out   : out flit_data;
67
68        south_rh_out    : out std_logic;
69        south_ri_out    : out std_logic;
70        south_re_out    : out std_logic;
71        south_ack_out   : in  std_logic;
72        south_data_out  : out flit_data;
73
74        east_rh_out     : out std_logic;
75        east_ri_out     : out std_logic;
76        east_re_out     : out std_logic;
77        east_ack_out    : in  std_logic;
78        east_data_out   : out flit_data;
79
80        local_rh_out    : out std_logic;
81        local_ri_out    : out std_logic;
82        local_re_out    : out std_logic;
83        local_ack_out   : in  std_logic;
84        local_data_out  : out flit_data
85      );
86    end component;
87
88    component traffic_source
89      generic(
90        ROM : rom_type := ROM_ZERO
91      );
92      port(
```

```
93        reset     : in  std_logic;
94        rh_out    : out STD_LOGIC;
95        ri_out    : out STD_LOGIC;
96        re_out    : out STD_LOGIC;
97        ack_out   : in  STD_LOGIC;
98        data_out  : out flit_data
99      );
100   end component;
101
102   component traffic_sink
103   port(
104     reset     : in  std_logic;
105     rh_in     : in  STD_LOGIC;
106     ri_in     : in  STD_LOGIC;
107     re_in     : in  STD_LOGIC;
108     ack_in    : out STD_LOGIC;
109     alive     : out std_logic;
110
111     -- ILA Signals --
112     ILA_clk   : out std_logic
113   );
114   end component;
115
116   ------------------------------------------------------------------
117   --
118   --  ICON core component declaration
119   --
120   ------------------------------------------------------------------
121   component icon
122     port
123     (
124       control0    :   out std_logic_vector(35 downto 0);
125       control1    :   out std_logic_vector(35 downto 0);
126       control2    :   out std_logic_vector(35 downto 0);
127       control3    :   out std_logic_vector(35 downto 0);
128       control4    :   out std_logic_vector(35 downto 0)
129     );
130   end component;
131
132   ------------------------------------------------------------------
133   --
134   --  ILA core component declaration
135   --
136   ------------------------------------------------------------------
137   component ila
138     port
139     (
140       control     : in    std_logic_vector(35 downto 0);
141       clk         : in    std_logic;
142       trig0       : in    std_logic_vector(15 downto 0)
143     );
144   end component;
145
146   component BUFG
147     port (
148       O : out STD_ULOGIC;
149       I : in STD_ULOGIC
150     );
151   end component;
152
153   signal rst : std_logic;
154
155   -- Handshake signals
156   signal north_rh_in,   north_ri_in,  north_re_in,  north_ack_in,
157          east_rh_in,    east_ri_in,   east_re_in,   east_ack_in,
```

```
158           south_rh_in,   south_ri_in,  south_re_in,  south_ack_in,
159           west_rh_in,    west_ri_in,   west_re_in,   west_ack_in,
160           local_rh_in,   local_ri_in,  local_re_in,  local_ack_in,
161           north_rh_out,  north_ri_out, north_re_out, north_ack_out,
162           east_rh_out,   east_ri_out,  east_re_out,  east_ack_out,
163           south_rh_out,  south_ri_out, south_re_out, south_ack_out,
164           west_rh_out,   west_ri_out,  west_re_out,  west_ack_out,
165           local_rh_out,  local_ri_out, local_re_out, local_ack_out   :
                  std_logic;
166
167     -- Data signals
168     signal north_data_in,   east_data_in,   south_data_in,  west_data_in,
            local_data_in,
169           north_data_out,  east_data_out,  south_data_out, west_data_out,
                  local_data_out : flit_data;
170
171
172     -- Chipscope signals
173     signal  north_sink_ila_control, east_sink_ila_control,
            south_sink_ila_control,
174           west_sink_ila_control,  local_sink_ila_control,
                  north_source_ila_control,
175           east_source_ila_control, south_source_ila_control,
                  west_source_ila_control, local_source_ila_control  :
                  std_logic_vector(35 downto 0);
176
177     signal north_ila_clk, east_ila_clk, south_ila_clk, west_ila_clk,
            local_ila_clk,north_req, east_req, south_req, west_req, local_req,
178           north_ila_clk_buf, east_ila_clk_buf, south_ila_clk_buf,
                  west_ila_clk_buf, local_ila_clk_buf   : std_logic;
179
180     attribute keep : string;
181     attribute keep of north_rh_in,  north_ri_in,  north_re_in,  north_ack_in,
182                       east_rh_in,     east_ri_in,   east_re_in,   east_ack_in,
183                       south_rh_in,  south_ri_in,  south_re_in,  south_ack_in,
184                       west_rh_in,     west_ri_in,   west_re_in,   west_ack_in,
185                       local_rh_in,  local_ri_in,  local_re_in,  local_ack_in,
186                       north_rh_out,   north_ri_out, north_re_out, north_ack_out
                              ,
187                       east_rh_out,  east_ri_out,  east_re_out,  east_ack_out,
188                       south_rh_out,   south_ri_out, south_re_out, south_ack_out
                              ,
189                       west_rh_out,  west_ri_out,  west_re_out,  west_ack_out,
190                       local_rh_out,   local_ri_out, local_re_out, local_ack_out
                                 : signal is "true";
191
192     attribute keep of north_data_in,  east_data_in,   south_data_in,
            west_data_in,   local_data_in,
193                       north_data_out,   east_data_out,  south_data_out,
                              west_data_out,  local_data_out : signal is "true";
194
195
196     attribute keep of north_sink_ila_control, east_sink_ila_control,
            south_sink_ila_control,
197                       west_sink_ila_control,  local_sink_ila_control,
198                       north_ila_clk, east_ila_clk, south_ila_clk, west_ila_clk,
                               local_ila_clk
199                        : signal is "true";
200   begin
201
202     BUFG_reset : BUFG
203     port map (
204       O => rst,
205       I => reset
206     );
```

```
207
208   BUFG_north_ila_clk : BUFG
209   port map (
210     O => north_ila_clk_buf, -- Clock buffer output
211     I => north_ila_clk      -- Clock buffer input
212   );
213
214   BUFG_east_ila_clk : BUFG
215   port map (
216     O => east_ila_clk_buf, -- Clock buffer output
217     I => east_ila_clk       -- Clock buffer input
218   );
219
220   BUFG_south_ila_clk : BUFG
221   port map (
222     O => south_ila_clk_buf, -- Clock buffer output
223     I => south_ila_clk      -- Clock buffer input
224   );
225
226   BUFG_west_ila_clk : BUFG
227   port map (
228     O => west_ila_clk_buf, -- Clock buffer output
229     I => west_ila_clk       -- Clock buffer input
230   );
231
232   BUFG_local_ila_clk : BUFG
233   port map (
234     O => local_ila_clk_buf, -- Clock buffer output
235     I => local_ila_clk      -- Clock buffer input
236   );
237
238   ------------------------------------------------------------------
239   --
240   --  ICON core instance
241   --
242   ------------------------------------------------------------------
243   i_icon : icon
244     port map
245     (
246       control0    => north_sink_ila_control,
247       control1    => east_sink_ila_control,
248       control2    => south_sink_ila_control,
249       control3    => west_sink_ila_control,
250       control4    => local_sink_ila_control
251     );
252
253   --- SOURCES --
254
255   north_source: traffic_source
256   generic map(
257     ROM => north_source_rom_data
258   )
259   port map(
260     reset     => rst,
261     rh_out    => north_rh_in,
262     ri_out    => north_ri_in,
263     re_out    => north_re_in,
264     ack_out   => north_ack_in,
265     data_out  => north_data_in
266   );
267
268   east_source: traffic_source
269   generic map(
270     ROM => east_source_rom_data
271   )
```

```
272   port map(
273     reset     => rst,
274     rh_out    => east_rh_in,
275     ri_out    => east_ri_in,
276     re_out    => east_re_in,
277     ack_out   => east_ack_in,
278     data_out  => east_data_in
279   );
280
281   south_source: traffic_source
282   generic map(
283     ROM => south_source_rom_data
284   )
285   port map(
286     reset     => rst,
287     rh_out    => south_rh_in,
288     ri_out    => south_ri_in,
289     re_out    => south_re_in,
290     ack_out   => south_ack_in,
291     data_out  => south_data_in
292   );
293
294   west_source: traffic_source
295   generic map(
296     ROM => west_source_rom_data
297   )
298   port map(
299     reset     => rst,
300     rh_out    => west_rh_in,
301     ri_out    => west_ri_in,
302     re_out    => west_re_in,
303     ack_out   => west_ack_in,
304     data_out  => west_data_in
305   );
306
307   local_source: traffic_source
308   generic map(
309     ROM => local_source_rom_data
310   )
311   port map(
312     reset     => rst,
313     rh_out    => local_rh_in,
314     ri_out    => local_ri_in,
315     re_out    => local_re_in,
316     ack_out   => local_ack_in,
317     data_out  => local_data_in
318   );
319
320
321   -- SINKS --
322
323   north_sink : traffic_sink
324   port map(
325     reset       => rst,
326     rh_in       => north_rh_out,
327     ri_in       => north_ri_out,
328     re_in       => north_re_out,
329     ack_in      => north_ack_out,
330     alive       => north_alive,
331     ILA_clk     => north_ila_clk
332
333   );
334
335   north_sink_ila : ila
336   port map
```

```
337     (
338       control   => north_sink_ila_control,
339       clk       => north_ila_clk_buf,
340       trig0     => north_data_out
341     );
342
343     east_sink : traffic_sink
344     port map(
345       reset       => rst,
346       rh_in       => east_rh_out,
347       ri_in       => east_ri_out,
348       re_in       => east_re_out,
349       ack_in      => east_ack_out,
350       alive       => east_alive,
351       ILA_clk     => east_ila_clk
352     );
353
354     east_sink_ila : ila
355     port map
356     (
357       control   => east_sink_ila_control,
358       clk       => east_ila_clk_buf,
359       trig0     => east_data_out
360     );
361
362     south_sink : traffic_sink
363     port map(
364       reset       => rst,
365       rh_in       => south_rh_out,
366       ri_in       => south_ri_out,
367       re_in       => south_re_out,
368       ack_in      => south_ack_out,
369       alive       => south_alive,
370       ILA_clk     => south_ila_clk
371     );
372
373     south_sink_ila : ila
374     port map
375     (
376       control   => south_sink_ila_control,
377       clk       => south_ila_clk_buf,
378       trig0     => south_data_out
379     );
380
381     west_sink : traffic_sink
382     port map(
383       reset       => rst,
384       rh_in       => west_rh_out,
385       ri_in       => west_ri_out,
386       re_in       => west_re_out,
387       ack_in      => west_ack_out,
388       alive       => west_alive,
389       ILA_clk     => west_ila_clk
390     );
391
392     west_sink_ila : ila
393     port map
394     (
395       control   => west_sink_ila_control,
396       clk       => west_ila_clk_buf,
397       trig0     => west_data_out
398     );
399
400     local_sink : traffic_sink
401     port map(
```

```
402       reset        => rst,
403       rh_in        => local_rh_out,
404       ri_in        => local_ri_out,
405       re_in        => local_re_out,
406       ack_in       => local_ack_out,
407       alive        => local_alive,
408       ILA_clk      => local_ila_clk
409     );
410
411     local_sink_ila : ila
412     port map
413     (
414       control   => local_sink_ila_control,
415       clk       => local_ila_clk_buf,
416       trig0     => local_data_out
417     );
418
419
420     router: be_router PORT MAP(
421       reset           => rst,
422       north_rh_in     => north_rh_in,
423       north_ri_in     => north_ri_in,
424       north_re_in     => north_re_in,
425       north_ack_in    => north_ack_in,
426       north_data_in   => north_data_in,
427       west_rh_in      => west_rh_in,
428       west_ri_in      => west_ri_in,
429       west_re_in      => west_re_in,
430       west_ack_in     => west_ack_in,
431       west_data_in    => west_data_in,
432       south_rh_in     => south_rh_in,
433       south_ri_in     => south_ri_in,
434       south_re_in     => south_re_in,
435       south_ack_in    => south_ack_in,
436       south_data_in   => south_data_in,
437       east_rh_in      => east_rh_in,
438       east_ri_in      => east_ri_in,
439       east_re_in      => east_re_in,
440       east_ack_in     => east_ack_in,
441       east_data_in    => east_data_in,
442       local_rh_in     => local_rh_in,
443       local_ri_in     => local_ri_in,
444       local_re_in     => local_re_in,
445       local_ack_in    => local_ack_in,
446       local_data_in   => local_data_in,
447       north_rh_out    => north_rh_out,
448       north_ri_out    => north_ri_out,
449       north_re_out    => north_re_out,
450       north_ack_out   => north_ack_out,
451       north_data_out  => north_data_out,
452       west_rh_out     => west_rh_out,
453       west_ri_out     => west_ri_out,
454       west_re_out     => west_re_out,
455       west_ack_out    => west_ack_out,
456       west_data_out   => west_data_out,
457       south_rh_out    => south_rh_out,
458       south_ri_out    => south_ri_out,
459       south_re_out    => south_re_out,
460       south_ack_out   => south_ack_out,
461       south_data_out  => south_data_out,
462       east_rh_out     => east_rh_out,
463       east_ri_out     => east_ri_out,
464       east_re_out     => east_re_out,
465       east_ack_out    => east_ack_out,
466       east_data_out   => east_data_out,
```

```
467       local_rh_out    => local_rh_out,
468       local_ri_out    => local_ri_out,
469       local_re_out    => local_re_out,
470       local_ack_out   => local_ack_out,
471       local_data_out  => local_data_out
472     );
473
474   end Behavioral;
```

A.5.3 Network Adapter

A.5.3.1 master_na.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   entity master_na is
8     generic(
9       routing_table : route_lookup_table_type
10    );
11    port(
12      clk_i   : in std_logic;
13      reset_i : in std_logic;
14
15      -- OCP interface
16      ocp_MCmd_i          : in  MCmdEncoding;
17      ocp_Maddr_i         : in  std_logic_vector(addr_width-1 downto 0);
18      ocp_MData_i         : in  std_logic_vector(addr_width-1 downto 0);
19      ocp_MByteEn_i       : in  std_logic_vector(3 downto 0);
20      ocp_SCmdAccept_o    : out std_logic;
21      ocp_SResp_o         : out SRespEncoding;
22      ocp_SData_o         : out std_logic_vector(addr_width-1 downto 0);
23
24      -- transmit hs channel
25      rh_out    : out std_logic;
26      ri_out    : out std_logic;
27      re_out    : out std_logic;
28      ack_out   : in  std_logic;
29      data_out  : out flit_data;
30
31      -- receive hs channel
32      rh_in   : in  std_logic;
33      ri_in   : in  std_logic;
34      re_in   : in  std_logic;
35      ack_in  : out std_logic;
36      data_in : in  flit_data
37
38    );
39  end master_na;
40
41  architecture Behavioral of master_na is
42
43    component ocp_master_transfer_unit is
44    generic(
45      routing_table : route_lookup_table_type
46    );
47    port(
48      clk_i               : in std_logic;
```

```
49      reset_i             : in std_logic;
50
51      -- OCP Interface
52      ocp_MCmd_i          : in  MCmdEncoding;
53      ocp_Maddr_i         : in  std_logic_vector(addr_width-1 downto 0);
54      ocp_MData_i         : in  std_logic_vector(addr_width-1 downto 0);
55      ocp_MByteEn_i       : in  std_logic_vector(3 downto 0);
56      ocp_SCmdAccept_o    : out std_logic;
57
58      -- Async inferface
59      sync_req_out            : out std_logic;
60      sync_ack_out            : in  std_logic;
61      packet_type_out         : out std_logic;
62      header_flit_data        : out flit_data;
63      control_flit_data       : out flit_data;
64      addr_flit_data          : out flit_data;
65      data_flit_data          : out flit_data
66
67    );
68    end component;
69
70    component ocp_master_receive_unit is
71    port(
72      clk_i               : in std_logic;
73      reset_i             : in std_logic;
74
75      -- OCP Interface
76      ocp_SResp_o         : out SRespEncoding;
77      ocp_SData_o         : out std_logic_vector(addr_width-1 downto 0);
78
79      -- Async inferface
80      sync_req_in           : in std_logic;
81      sync_ack_in           : out  std_logic;
82      sresp_flit            : in  flit_data;
83      sdata_flit            : in  flit_data
84    );
85    end component;
86
87    component synchronizer is
88    port(
89      clk_in    : in  std_logic;
90      reset_i   : in  std_logic;
91      async_in  : in  std_logic;
92      sync_out  : out std_logic
93    );
94    end component;
95
96    component async_transmitter is
97    port(
98      reset           : in  std_logic;
99      sync_req_in     : in  std_logic;
100     sync_ack_in     : out std_logic;
101     packet_type_in  : in  std_logic;
102     header_flit_in  : in  flit_data;
103     control_flit_in : in  flit_data;
104     addr_flit_in    : in  flit_data;
105     data_flit_in    : in  flit_data;
106     rh_out          : out std_logic;
107     ri_out          : out std_logic;
108     re_out          : out std_logic;
109     ack_out         : in  std_logic;
110     data_out        : out flit_data
111
112   );
113   end component;
```

```
114
115     component async_receiver is
116     port(
117       reset             : in  std_logic;
118       rh_in             : in  std_logic;
119       ri_in             : in  std_logic;
120       re_in             : in  std_logic;
121       ack_in            : out std_logic;
122       data_in           : in  flit_data;
123       sync_req_out      : out std_logic;
124       sync_ack_out      : in  std_logic;
125       header_flit_out   : out flit_data;
126       control_flit_out  : out flit_data;
127       ad_flit0_out      : out flit_data;
128       ad_flit1_out      : out flit_data
129     );
130     end component;
131
132     signal transmit_req,transmit_ack,transmit_ack_sync,transmit_packet_type,
          receive_req, receive_req_sync, receive_ack : std_logic;
133
134     signal transmit_header_flit, transmit_control_flit, transmit_addr_flit,
          transmit_data_flit, receive_sresp_flit, receive_sdata_flit : flit_data
          ;
135
136     attribute keep : string;
137     attribute keep of transmit_req, receive_ack  : signal is "true";    --so we
           can put TIG on them in UCF
138
139   begin
140
141     ocp_transfer : ocp_master_transfer_unit
142     generic map(
143       routing_table       => routing_table
144     )
145     port map(
146       clk_i               => clk_i,
147       reset_i             => reset_i,
148
149       -- OCP Interface
150       ocp_MCmd_i          => ocp_MCmd_i,
151       ocp_Maddr_i         => ocp_Maddr_i,
152       ocp_MData_i         => ocp_MData_i,
153       ocp_MByteEn_i       => ocp_MByteEn_i,
154       ocp_SCmdAccept_o    => ocp_SCmdAccept_o,
155
156       -- Async inferface
157       sync_req_out        => transmit_req,
158       sync_ack_out        => transmit_ack_sync,
159       packet_type_out     => transmit_packet_type,
160       header_flit_data    => transmit_header_flit,
161       control_flit_data   => transmit_control_flit,
162       addr_flit_data      => transmit_addr_flit,
163       data_flit_data      => transmit_data_flit
164     );
165
166     async_transmit : async_transmitter
167     port map(
168       reset             => reset_i,
169       sync_req_in       => transmit_req,
170       sync_ack_in       => transmit_ack,
171       packet_type_in    => transmit_packet_type,
172       header_flit_in    => transmit_header_flit,
173       control_flit_in   => transmit_control_flit,
174       addr_flit_in      => transmit_addr_flit,
```

```
175       data_flit_in      => transmit_data_flit,
176       rh_out            => rh_out,
177       ri_out            => ri_out,
178       re_out            => re_out,
179       ack_out           => ack_out,
180       data_out          => data_out
181     );
182
183     transmit_sync : synchronizer
184     port map(
185       clk_in    => clk_i,
186       reset_i   => reset_i,
187       async_in  => transmit_ack,
188       sync_out  => transmit_ack_sync
189     );
190
191     ocp_receive : ocp_master_receive_unit
192     port map(
193       clk_i               => clk_i,
194       reset_i             => reset_i,
195
196       -- OCP Interface
197       ocp_SResp_o         => ocp_SResp_o,
198       ocp_SData_o         => ocp_SData_o,
199
200       -- Async inferface
201       sync_req_in         => receive_req_sync,
202       sync_ack_in         => receive_ack,
203       sresp_flit          => receive_sresp_flit,
204       sdata_flit          => receive_sdata_flit
205     );
206
207     async_receive : async_receiver
208     port map(
209       reset             => reset_i,
210       rh_in             => rh_in,
211       ri_in             => ri_in,
212       re_in             => re_in,
213       ack_in            => ack_in,
214       data_in           => data_in,
215       sync_req_out      => receive_req,
216       sync_ack_out      => receive_ack,
217       header_flit_out   => open,
218       control_flit_out  => receive_sresp_flit,
219       ad_flit0_out      => open,
220       ad_flit1_out      => receive_sdata_flit
221     );
222
223     receive_sync : synchronizer
224     port map(
225       clk_in    => clk_i,
226       reset_i   => reset_i,
227       async_in  => receive_req,
228       sync_out  => receive_req_sync
229     );
230
231   end Behavioral;
```

A.5.3.2 ocp_master_transfer_unit.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
```

```
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity ocp_master_transfer_unit is
8    generic(
9      routing_table : route_lookup_table_type
10   );
11   port(
12     clk_i               : in std_logic;
13     reset_i             : in std_logic;
14
15     -- OCP Interface
16     ocp_MCmd_i          : in  MCmdEncoding;
17     ocp_Maddr_i         : in  std_logic_vector(addr_width-1 downto 0);
18     ocp_MData_i         : in  std_logic_vector(addr_width-1 downto 0);
19     ocp_MByteEn_i       : in  std_logic_vector(3 downto 0);
20     ocp_SCmdAccept_o    : out std_logic;
21
22     -- Async inferface
23     sync_req_out            : out std_logic;
24     sync_ack_out            : in  std_logic;
25     packet_type_out         : out std_logic;
26     header_flit_data        : out flit_data;
27     control_flit_data       : out flit_data;
28     addr_flit_data          : out flit_data;
29     data_flit_data          : out flit_data
30
31   );
32 end ocp_master_transfer_unit;
33
34 architecture Behavioral of ocp_master_transfer_unit is
35   type state is (WAIT_CMD, STORE_PACKET, ROUTE_LOOKUP, REQUEST, ACKNOWLEDGE);
36
37   signal current_state, next_state : state;
38   signal MCmd  : MCmdEncoding;
39   signal MAddr, MData : std_logic_vector(addr_width-1 downto 0);
40   signal MByteEn : std_logic_vector(3 downto 0);
41
42   constant control_flit_zero_part : std_logic_vector(FLIT_SIZE-1 downto 7) :=
         (others => '0');
43
44   signal ocp_register_en, sync_req : std_logic;
45
46   signal reverse_path : flit_data;
47
48   signal route_lookup_value : std_logic_vector(2*FLIT_SIZE-1 downto 0);
49
50 begin
51
52   next_state_logic : process(current_state, ocp_MCmd_i, sync_ack_out)
53   begin
54     case current_state is
55       when WAIT_CMD =>
56         ocp_register_en   <= '0';
57         ocp_SCmdAccept_o  <= '0';
58         sync_req      <= '0';
59
60         if ocp_MCmd_i /= MCmd_IDLE then
61           next_state <= STORE_PACKET;
62         else
63           next_state <= WAIT_CMD;
64         end if;
65       when STORE_PACKET =>
66         ocp_register_en   <= '1';
67         ocp_SCmdAccept_o  <= '0';
```

```
68          sync_req      <= '0';
69
70          next_state <= ROUTE_LOOKUP;
71        when ROUTE_LOOKUP =>
72          ocp_register_en   <= '0';
73          ocp_SCmdAccept_o  <= '1';
74          sync_req      <= '0';
75
76          next_state <= REQUEST;
77        when REQUEST =>
78          ocp_register_en   <= '0';
79          ocp_SCmdAccept_o  <= '0';
80          sync_req      <= '1';
81
82          if sync_ack_out = '1' then
83            next_state <= ACKNOWLEDGE;
84          else
85            next_state <= REQUEST;
86          end if;
87        when ACKNOWLEDGE =>
88          ocp_register_en   <= '0';
89          ocp_SCmdAccept_o  <= '0';
90          sync_req      <= '0';
91
92          if sync_ack_out = '0' then
93            next_state <= WAIT_CMD;
94          else
95            next_state <= ACKNOWLEDGE;
96          end if;
97      end case;
98    end process;
99
100   state_register : process(clk_i, reset_i)
101   begin
102     if reset_i = '0' then   --active low
103       current_state <= WAIT_CMD;
104     elsif rising_edge(clk_i) then
105       current_state <= next_state;
106     end if;
107   end process;
108
109   ocp_cmd_register : process (clk_i,ocp_register_en)
110   begin
111     if rising_edge(clk_i) then
112       if ocp_register_en = '1' then
113         MCmd      <= ocp_MCmd_i;
114         MAddr     <= ocp_Maddr_i;
115         MData     <= ocp_Mdata_i;
116         MByteEn   <= ocp_MByteEn_i;
117       end if;
118     end if;
119   end process;
120
121   route_lookup_process : process(clk_i,ocp_Maddr_i)
122   begin
123     if rising_edge(clk_i) then
124       route_lookup_value <= routing_table(conv_integer(Maddr(addr_width-1
              downto addr_width-4)));
125     end if;
126   end process;
127
128   --forward path
129   header_flit_data <= route_lookup_value(2*FLIT_SIZE-1 downto FLIT_SIZE);
130   --return path
131   reverse_path     <= route_lookup_value(FLIT_SIZE-1 downto 0);
```

```
132
133   -- Control flit
134   control_flit_data <= control_flit_zero_part & MCmd & MByteEn;
135   -- Address flit
136   addr_flit_data <= MAddr;
137   -- reverse_path (RD) or data flit (WR)
138   data_flit_data <= reverse_path when MCmd = MCmd_RD else
139                     MData;
140
141   packet_type_out <= '1' when MCmd = MCmd_WR else
142                      '0';
143
144   -- deglitch ff on sync_req
145   deglitch : process(clk_i,reset_i,sync_req)
146   begin
147     if reset_i = '0' then
148       sync_req_out <= '0';
149     elsif rising_edge(clk_i) then
150       sync_req_out <= sync_req;
151     end if;
152   end process;
153
154 end Behavioral;
```

A.5.3.3 ocp_master_receive_unit.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity ocp_master_receive_unit is
8    port(
9      clk_i               : in std_logic;
10     reset_i             : in std_logic;
11
12     -- OCP Interface
13     ocp_SResp_o         : out SRespEncoding;
14     ocp_SData_o         : out std_logic_vector(addr_width-1 downto 0);
15
16     -- Async inferface
17     sync_req_in           : in std_logic;
18     sync_ack_in           : out  std_logic;
19     sresp_flit            : in  flit_data;
20     sdata_flit            : in  flit_data
21   );
22 end ocp_master_receive_unit;
23
24 architecture Behavioral of ocp_master_receive_unit is
25   type state is (WAIT_REQ, STORE_PACKET, ACKNOWLEDGE);
26
27   signal current_state, next_state : state;
28
29   signal ocp_register_en, sync_ack : std_logic;
30
31 begin
32
33   next_state_logic : process(current_state, sync_req_in)
34   begin
35     case current_state is
36       when WAIT_REQ =>
37         sync_ack    <= '0';
```

```
38          ocp_register_en <= '0';
39
40          if sync_req_in = '1' then
41            next_state <= STORE_PACKET;
42          else
43            next_state <= WAIT_REQ;
44          end if;
45        when STORE_PACKET =>
46          sync_ack    <= '0';
47          ocp_register_en <= '1';
48
49          next_state <= ACKNOWLEDGE;
50        when ACKNOWLEDGE =>
51          sync_ack    <= '1';
52          ocp_register_en <= '0';
53
54          if sync_req_in = '0' then
55            next_state <= WAIT_REQ;
56          else
57            next_state <= ACKNOWLEDGE;
58          end if;
59        end case;
60    end process;
61
62    state_register : process(clk_i, reset_i,next_state)
63    begin
64      if reset_i = '0' then   --active low
65        current_state <= WAIT_REQ;
66      elsif rising_edge(clk_i) then
67        current_state <= next_state;
68      end if;
69    end process;
70
71    ocp_cmd_register : process (ocp_register_en)
72    begin
73      if ocp_register_en = '1' then
74        ocp_SResp_o <= sresp_flit(1 downto 0);
75        ocp_SData_o <= sdata_flit;
76      else
77        ocp_SResp_o <= (others => '0');
78        ocp_SData_o <= (others => '0');
79      end if;
80    end process;
81
82    -- deglitch ff on sync_ack
83    deglitch : process(clk_i,reset_i,sync_ack)
84    begin
85      if reset_i = '0' then
86        sync_ack_in <= '0';
87      elsif rising_edge(clk_i) then
88        sync_ack_in <= sync_ack;
89      end if;
90    end process;
91
92  end Behavioral;
```

A.5.3.4 slave_na.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
```

```
6
7  entity slave_na is
8    port(
9      clk_i   : in std_logic;
10     reset_i : in std_logic;
11
12     -- OCP interface
13     ocp_MCmd_o          : out MCmdEncoding;
14     ocp_Maddr_o         : out std_logic_vector(addr_width-1 downto 0);
15     ocp_MData_o         : out std_logic_vector(addr_width-1 downto 0);
16     ocp_MByteEn_o       : out std_logic_vector(3 downto 0);
17     ocp_SCmdAccept_i    : in  std_logic;
18     ocp_SResp_i         : in  SRespEncoding;
19     ocp_SData_i         : in  std_logic_vector(addr_width-1 downto 0);
20
21     -- transmit hs channel
22     rh_out    : out std_logic;
23     ri_out    : out std_logic;
24     re_out    : out std_logic;
25     ack_out   : in  std_logic;
26     data_out  : out flit_data;
27
28     -- receive hs channel
29     rh_in   : in  std_logic;
30     ri_in   : in  std_logic;
31     re_in   : in  std_logic;
32     ack_in  : out std_logic;
33     data_in : in  flit_data
34
35   );
36 end slave_na;
37
38 architecture Behavioral of slave_na is
39
40   component ocp_slave_receive_unit is
41   port(
42     clk_i               : in std_logic;
43     reset_i             : in std_logic;
44
45     -- OCP Interface
46     ocp_MCmd_o          : out MCmdEncoding;
47     ocp_Maddr_o         : out std_logic_vector(addr_width-1 downto 0);
48     ocp_MData_o         : out std_logic_vector(addr_width-1 downto 0);
49     ocp_MByteEn_o       : out std_logic_vector(3 downto 0);
50     ocp_SCmdAccept_i    : in std_logic;
51
52     -- Async inferface
53     sync_req_in           : in std_logic;
54     sync_ack_in           : out  std_logic;
55     header_flit_in        : in  flit_data;
56     control_flit_in       : in  flit_data;
57     addr_flit_in          : in  flit_data;
58     data_flit_in          : in  flit_data;
59
60     --control
61     read_cmd_out    : out std_logic;
62     read_cmd_done   : in std_logic;
63     reverse_header  : out flit_data
64   );
65   end component;
66
67   component ocp_slave_transfer_unit is
68   port(
69     clk_i               : in std_logic;
70     reset_i             : in std_logic;
```

```
71
72      -- OCP Interface
73      ocp_SResp_i          : in SRespEncoding;
74      ocp_SData_i          : in std_logic_vector(addr_width-1 downto 0);
75
76      -- Async inferface
77      sync_req_out           : out std_logic;
78      sync_ack_out           : in  std_logic;
79      header_flit_out        : out flit_data;
80      control_flit_out       : out flit_data;
81      data_flit_out          : out flit_data;
82
83      --Control
84      read_cmd_in        : in std_logic;
85      read_cmd_done      : out std_logic;
86      header_flit_in     : in flit_data
87
88    );
89    end component;
90
91    component synchronizer is
92    port(
93      clk_in    : in  std_logic;
94      reset_i   : in  std_logic;
95      async_in  : in  std_logic;
96      sync_out  : out std_logic
97    );
98    end component;
99
100   component async_transmitter is
101   port(
102     reset          : in  std_logic;
103     sync_req_in    : in  std_logic;
104     sync_ack_in    : out std_logic;
105     packet_type_in     : in  std_logic;
106     header_flit_in     : in  flit_data;
107     control_flit_in : in  flit_data;
108     addr_flit_in       : in  flit_data;
109     data_flit_in       : in  flit_data;
110     rh_out         : out std_logic;
111     ri_out         : out std_logic;
112     re_out         : out std_logic;
113     ack_out        : in  std_logic;
114     data_out       : out flit_data
115
116   );
117   end component;
118
119   component async_receiver is
120   port(
121     reset              : in  std_logic;
122     rh_in              : in  std_logic;
123     ri_in              : in  std_logic;
124     re_in              : in  std_logic;
125     ack_in             : out std_logic;
126     data_in            : in  flit_data;
127     sync_req_out       : out std_logic;
128     sync_ack_out       : in  std_logic;
129     header_flit_out    : out flit_data;
130     control_flit_out   : out flit_data;
131     ad_flit0_out       : out flit_data;
132     ad_flit1_out       : out flit_data
133   );
134   end component;
135
```

```
136   signal transmit_req,transmit_ack,transmit_ack_sync,transmit_packet_type,
          receive_req, receive_req_sync, receive_ack,
137         read_cmd, read_cmd_done : std_logic;
138
139   signal transmit_header_flit, transmit_control_flit, transmit_addr_flit,
          transmit_data_flit,
140         receive_header_flit,  receive_control_flit,  receive_addr_flit,
                receive_data_flit,reversed_header : flit_data;
141
142   attribute keep : string;
143   attribute keep of transmit_req, receive_ack  : signal is "true";    --so we
          can put TIG on them in UCF
144
145 begin
146
147   ocp_receive : ocp_slave_receive_unit
148   port map (
149     clk_i               => clk_i,
150     reset_i             => reset_i,
151
152     -- OCP Interface
153     ocp_MCmd_o          => ocp_MCmd_o,
154     ocp_Maddr_o         => ocp_Maddr_o,
155     ocp_MData_o         => ocp_MData_o,
156     ocp_MByteEn_o       => ocp_MByteEn_o,
157     ocp_SCmdAccept_i    => ocp_SCmdAccept_i,
158
159     -- Async inferface
160     sync_req_in         => receive_req_sync,
161     sync_ack_in         => receive_ack,
162     header_flit_in      => receive_header_flit,
163     control_flit_in     => receive_control_flit,
164     addr_flit_in        => receive_addr_flit,
165     data_flit_in        => receive_data_flit,
166
167     --control
168     read_cmd_out        => read_cmd,
169     read_cmd_done       => read_cmd_done,
170     reverse_header      => reversed_header
171   );
172
173   ocp_transfer : ocp_slave_transfer_unit
174   port map (
175     clk_i               => clk_i,
176     reset_i             => reset_i,
177
178     -- OCP Interface
179     ocp_SResp_i         => ocp_SResp_i,
180     ocp_SData_i         => ocp_SData_i,
181
182     -- Async inferface
183     sync_req_out        => transmit_req,
184     sync_ack_out        => transmit_ack_sync,
185     header_flit_out     => transmit_header_flit,
186     control_flit_out    => transmit_control_flit,
187     data_flit_out       => transmit_data_flit,
188
189     --Control
190     read_cmd_in         => read_cmd,
191     read_cmd_done       => read_cmd_done,
192     header_flit_in      => reversed_header
193
194   );
195
196   async_transmit : async_transmitter
```

```
197   port map(
198     reset             => reset_i,
199     sync_req_in       => transmit_req,
200     sync_ack_in       => transmit_ack,
201     packet_type_in    => transmit_packet_type,
202     header_flit_in    => transmit_header_flit,
203     control_flit_in   => transmit_control_flit,
204     addr_flit_in      => (others => '0'),       -- no addr flit for read
          response packets
205     data_flit_in      => transmit_data_flit,
206     rh_out            => rh_out,
207     ri_out            => ri_out,
208     re_out            => re_out,
209     ack_out           => ack_out,
210     data_out          => data_out
211   );
212
213   transmit_sync : synchronizer
214   port map(
215     clk_in    => clk_i,
216     reset_i   => reset_i,
217     async_in  => transmit_ack,
218     sync_out  => transmit_ack_sync
219   );
220
221   async_receive : async_receiver
222   port map(
223     reset             => reset_i,
224     rh_in             => rh_in,
225     ri_in             => ri_in,
226     re_in             => re_in,
227     ack_in            => ack_in,
228     data_in           => data_in,
229     sync_req_out      => receive_req,
230     sync_ack_out      => receive_ack,
231     header_flit_out   => receive_header_flit,
232     control_flit_out  => receive_control_flit,
233     ad_flit0_out      => receive_addr_flit,
234     ad_flit1_out      => receive_data_flit
235   );
236
237   receive_sync : synchronizer
238   port map(
239     clk_in    => clk_i,
240     reset_i   => reset_i,
241     async_in  => receive_req,
242     sync_out  => receive_req_sync
243   );
244
245 end Behavioral;
```

A.5.3.5 ocp_slave_transfer_unit.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity ocp_slave_transfer_unit is
8    port(
9      clk_i               : in std_logic;
10     reset_i             : in std_logic;
```

```
11
12      -- OCP Interface
13      ocp_SResp_i          : in SRespEncoding;
14      ocp_SData_i          : in std_logic_vector(addr_width-1 downto 0);
15
16      -- Async inferface
17      sync_req_out           : out std_logic;
18      sync_ack_out           : in  std_logic;
19      header_flit_out        : out flit_data;
20      control_flit_out       : out flit_data;
21      data_flit_out          : out flit_data;
22
23      --Control
24      read_cmd_in        : in std_logic;
25      read_cmd_done      : out std_logic;
26      header_flit_in     : in flit_data
27
28    );
29  end ocp_slave_transfer_unit;
30
31  architecture Behavioral of ocp_slave_transfer_unit is
32
33    type state is (INIT, WAIT_SRESP, STORE_DATA, REQUEST, ACKNOWLEDGE,
          WAIT_READ_CMD_DONE);
34    signal current_state, next_state : state;
35    constant control_flit_zero_part : std_logic_vector(FLIT_SIZE-1 downto 2) :=
          (others => '0');
36    signal ocp_register_en, sync_req : std_logic;
37
38  begin
39
40    next_state_logic : process(current_state, ocp_SResp_i, sync_ack_out,
          read_cmd_in)
41    begin
42      case current_state is
43        when INIT =>
44          sync_req <= '0';
45          read_cmd_done <= '0';
46          ocp_register_en <= '0';
47
48          if read_cmd_in = '1' then
49            next_state <= WAIT_SRESP;
50          else
51            next_state <= INIT;
52          end if;
53        when WAIT_SRESP =>
54          sync_req <= '0';
55
56          read_cmd_done <= '0';
57          if ocp_SResp_i = SResp_DVA then
58            ocp_register_en <= '1';
59            next_state <= REQUEST;
60          else
61            ocp_register_en <= '0';
62            next_state <= WAIT_SRESP;
63          end if;
64        when STORE_DATA =>
65          sync_req <= '0';
66          ocp_register_en <= '1';
67          read_cmd_done <= '0';
68          next_state <= REQUEST;
69        when REQUEST =>
70          sync_req <= '1';
71          ocp_register_en <= '0';
72          read_cmd_done <= '0';
```

```
73          if sync_ack_out = '1' then
74            next_state <= ACKNOWLEDGE;
75          else
76            next_state <= REQUEST;
77          end if;
78        when ACKNOWLEDGE =>
79          sync_req <= '0';
80          read_cmd_done <= '0';
81          ocp_register_en <= '0';
82          if sync_ack_out = '0' then
83            next_state <= WAIT_READ_CMD_DONE;
84          else
85            next_state <= ACKNOWLEDGE;
86          end if;
87        when WAIT_READ_CMD_DONE =>
88          sync_req <= '0';
89          read_cmd_done <= '1';
90          ocp_register_en <= '0';
91
92          if read_cmd_in = '0' then
93            next_state <= INIT;
94          else
95            next_state <= WAIT_READ_CMD_DONE;
96          end if;
97      end case;
98    end process;
99
100   state_register : process(clk_i, reset_i)
101   begin
102     if reset_i = '0' then   --active low
103       current_state <= INIT;
104     elsif rising_edge(clk_i) then
105       current_state <= next_state;
106     end if;
107   end process;
108
109   ocp_cmd_register : process (clk_i,ocp_register_en)
110   begin
111     if rising_edge(clk_i) then
112       if ocp_register_en = '1' then
113         control_flit_out  <= control_flit_zero_part & ocp_SResp_i;
114         data_flit_out     <= ocp_SData_i;
115       end if;
116     end if;
117   end process;
118
119   header_flit_out <= header_flit_in;
120
121   -- deglitch ff on sync_req
122   deglitch : process(clk_i,reset_i,sync_req)
123   begin
124     if reset_i = '0' then
125       sync_req_out <= '0';
126     elsif rising_edge(clk_i) then
127       sync_req_out <= sync_req;
128     end if;
129   end process;
130
131 end Behavioral;
```

A.5.3.6 ocp_slave_receive_unit.vhd

```
1  library IEEE;
```

```
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity ocp_slave_receive_unit is
8    port(
9      clk_i               : in std_logic;
10     reset_i             : in std_logic;
11
12     -- OCP Interface
13     ocp_MCmd_o          : out MCmdEncoding;
14     ocp_Maddr_o         : out std_logic_vector(addr_width-1 downto 0);
15     ocp_MData_o         : out std_logic_vector(addr_width-1 downto 0);
16     ocp_MByteEn_o       : out std_logic_vector(3 downto 0);
17     ocp_SCmdAccept_i    : in std_logic;
18
19     -- Async inferface
20     sync_req_in           : in  std_logic;
21     sync_ack_in           : out std_logic;
22     header_flit_in        : in  flit_data;
23     control_flit_in       : in  flit_data;
24     addr_flit_in          : in  flit_data;
25     data_flit_in          : in  flit_data;
26
27     --control
28     read_cmd_out    : out std_logic;
29     read_cmd_done   : in  std_logic;
30     reverse_header  : out flit_data
31   );
32 end ocp_slave_receive_unit;
33
34 architecture Behavioral of ocp_slave_receive_unit is
35
36   type state is (WAIT_REQ, STORE_PACKET, WAIT_CMDACCEPT, ACKNOWLEDGE,
         WAIT_READ_CMD_DONE);
37   signal current_state, next_state : state;
38   signal ocp_register_en, read_cmd, sync_ack : std_logic;
39
40 begin
41
42   read_cmd_out <= read_cmd;
43
44   next_state_logic : process(current_state, ocp_SCmdAccept_i, sync_req_in,
         control_flit_in, read_cmd_done)
45   begin
46     case current_state is
47       when WAIT_REQ =>
48         sync_ack        <= '0';
49         ocp_register_en <= '0';
50         read_cmd        <= '0';
51
52         if sync_req_in = '1' then
53           next_state <= STORE_PACKET;
54         else
55           next_state <= WAIT_REQ;
56         end if;
57       when STORE_PACKET =>
58          ocp_register_en <= '1';
59          sync_ack        <= '0';
60          if control_flit_in(6 downto 4) = MCmd_RD then    --MCmd field
61             read_cmd <= '1';
62          else
63             read_cmd <= '0';
64          end if;
```

```
65
66            --check SCmdAccept immediately
67            if ocp_SCmdAccept_i = '1' then
68               if control_flit_in(6 downto 4) = MCmd_RD then
69                 next_state <= WAIT_READ_CMD_DONE;
70               else
71                 next_state <= ACKNOWLEDGE;
72               end if;
73            else
74              next_state <= WAIT_CMDACCEPT;
75            end if;
76
77          when WAIT_CMDACCEPT =>
78            ocp_register_en <= '1';
79            sync_ack        <= '0';
80
81            if control_flit_in(6 downto 4) = MCmd_RD then    --MCmd field
82                read_cmd <= '1';
83             else
84                read_cmd <= '0';
85             end if;
86
87            if ocp_SCmdAccept_i = '1' then
88               if control_flit_in(6 downto 4) = MCmd_RD then
89                 next_state <= WAIT_READ_CMD_DONE;
90               else
91                 next_state <= ACKNOWLEDGE;
92               end if;
93             else
94               next_state <= WAIT_CMDACCEPT;
95             end if;
96          when WAIT_READ_CMD_DONE =>
97            sync_ack        <= '0';
98            ocp_register_en <= '0';
99            read_cmd        <= '1';
100
101           if read_cmd_done = '1' then
102             next_state <= ACKNOWLEDGE;
103           else
104             next_state <= WAIT_READ_CMD_DONE;
105           end if;
106
107         when ACKNOWLEDGE =>
108           ocp_register_en <= '1';
109           sync_ack        <= '1';
110           read_cmd        <= '0';
111
112           if sync_req_in = '0' then
113             next_state <= WAIT_REQ;
114           else
115             next_state <= ACKNOWLEDGE;
116           end if;
117       end case;
118   end process;
119
120   state_register : process(clk_i, reset_i,next_state)
121   begin
122     if reset_i = '0' then   --active low
123       current_state <= WAIT_REQ;
124     elsif rising_edge(clk_i) then
125       current_state <= next_state;
126     end if;
127   end process;
128
129   ocp_cmd_register : process (clk_i,ocp_register_en)
```

```
130   begin
131     if rising_edge(clk_i) then
132       if ocp_register_en = '1' then
133         ocp_MCmd_o    <= control_flit_in(6 downto 4);
134         ocp_MByteEn_o <= control_flit_in(3 downto 0);
135         ocp_Maddr_o   <= addr_flit_in;
136         ocp_MData_o   <= data_flit_in;
137       else
138         ocp_MCmd_o    <= (others => '0');   -- must be set to IDLE
139         ocp_MByteEn_o <= (others => '0');
140         ocp_Maddr_o   <= (others => '0');
141         ocp_MData_o   <= (others => '0');
142       end if;
143     end if;
144   end process;
145
146   -- If the packet is a RD request, store the reverse path
147   reverse_header_register : process(clk_i, read_cmd, data_flit_in)
148   begin
149     if rising_edge(clk_i) then
150       if read_cmd = '1' then
151         reverse_header <= data_flit_in;
152       end if;
153     end if;
154   end process;
155
156   -- deglitch ff on sync_ack_in - Glitch observed in post-par sim.
157   deglitch : process(clk_i,reset_i,sync_ack)
158   begin
159     if reset_i = '0' then
160       sync_ack_in <= '0';
161     elsif rising_edge(clk_i) then
162       sync_ack_in <= sync_ack;
163     end if;
164   end process;
165
166 end Behavioral;
```

A.5.3.7 async_transmitter.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity async_transmitter is
8    port(
9      reset           : in  std_logic;
10     sync_req_in     : in  std_logic;
11     sync_ack_in     : out std_logic;
12     packet_type_in  : in  std_logic;
13     header_flit_in  : in  flit_data;
14     control_flit_in : in  flit_data;
15     addr_flit_in    : in  flit_data;
16     data_flit_in    : in  flit_data;
17     rh_out          : out std_logic;
18     ri_out          : out std_logic;
19     re_out          : out std_logic;
20     ack_out         : in  std_logic;
21     data_out        : out flit_data
22
23   );
```

```
24
25  end async_transmitter;
26
27  architecture Behavioral of async_transmitter is
28
29    component async_transmitter_hs_ctrl is
30    port(
31      reset             : in  std_logic;
32      sync_req_in       : in  std_logic;
33      sync_ack_in       : out std_logic;
34      rh_out            : out std_logic;
35      ri_out            : out std_logic;
36      re_out            : out std_logic;
37      ack_out           : in  std_logic;
38      header_flit_out   : out std_logic;
39      control_flit_out  : out std_logic;
40      addr_flit_out     : out std_logic;
41      data_flit_out     : out std_logic
42    );
43    end component;
44
45    component as_bd_4p_delay is
46      generic(
47        size : natural := 10  -- Delay size
48      );
49      port (
50        d : in  std_logic;    -- Data in
51        z : out std_logic     -- Data out
52      );
53    end component;
54
55    signal rh, ri, re, header_flit_en, control_flit_en, addr_flit_en,
          data_flit_en  : std_logic;
56  begin
57
58    data_mux : process(header_flit_en,control_flit_en,addr_flit_en,
          data_flit_en, header_flit_in, control_flit_in, addr_flit_in,
          data_flit_in)
59    begin
60      if header_flit_en = '1' then
61        data_out <= header_flit_in;
62      elsif control_flit_en = '1' then
63        data_out <= control_flit_in;
64      elsif addr_flit_en = '1' then
65        data_out <= addr_flit_in;
66      elsif data_flit_en = '1' then
67        data_out <= data_flit_in;
68      else
69        data_out <= FLIT_ZERO;
70      end if;
71    end process;
72
73
74    hs_ctrl : async_transmitter_hs_ctrl
75    port map(
76      reset             => reset,
77      sync_req_in       => sync_req_in,
78      sync_ack_in       => sync_ack_in,
79      rh_out            => rh,
80      ri_out            => ri,
81      re_out            => re,
82      ack_out           => ack_out,
83      header_flit_out   => header_flit_en,
84      control_flit_out  => control_flit_en,
85      addr_flit_out     => addr_flit_en,
```

```
86      data_flit_out     => data_flit_en
87    );
88
89    rh_delay : as_bd_4p_delay
90    generic map(
91      size => 4
92    )
93    port map(
94      d => rh,
95      z => rh_out
96    );
97
98    ri_delay : as_bd_4p_delay
99    generic map(
100     size => 4
101   )
102   port map(
103     d => ri,
104     z => ri_out
105   );
106
107   re_delay : as_bd_4p_delay
108   generic map(
109     size => 4
110   )
111   port map(
112     d => re,
113     z => re_out
114   );
115
116 end Behavioral;
```

A.5.3.8 async_transmitter_hs_ctrl.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   library UNISIM;
7   use UNISIM.VComponents.lut2;
8   use UNISIM.VComponents.lut3;
9   use UNISIM.VComponents.lut4;
10  use UNISIM.VComponents.lut4_1;
11
12  entity async_transmitter_hs_ctrl is
13    port(
14      reset             : in  std_logic;
15      sync_req_in       : in  std_logic;
16      sync_ack_in       : out std_logic;
17      rh_out            : out std_logic;
18      ri_out            : out std_logic;
19      re_out            : out std_logic;
20      ack_out           : in  std_logic;
21      header_flit_out   : out std_logic;
22      control_flit_out  : out std_logic;
23      addr_flit_out     : out std_logic;
24      data_flit_out     : out std_logic
25    );
26  end async_transmitter_hs_ctrl;
27
28  architecture Behavioral of async_transmitter_hs_ctrl is
29
```

```
30    signal sync_req, ack, header_flit, rh, control_flit, ri, addr_flit,
          data_flit, sync_ack, re, csc0, csc1, csc2,
31            zero, one, four, five, eight, nine, eleven, twelve, fourteen,
                  seventeen, eighteen, twenty,
32            not_one, not_five, not_nine, not_twelve, not_csc1, not_eighteen,
                  not_twenty, not_sync_req, not_control_flit : std_logic;
33
34    attribute keep : string;
35    attribute keep of sync_req, ack, header_flit, rh, control_flit, ri,
          addr_flit, data_flit, sync_ack, re, csc0, csc1, csc2,
36            zero, one, four, five, eight, nine, eleven, twelve, fourteen,
                  seventeen, eighteen, twenty : signal is "true";
37
38    attribute rloc : string;
39    attribute rloc of zero_LUT          : label is "X0Y0";
40    attribute rloc of one_LUT           : label is "X0Y0";
41    attribute rloc of four_LUT          : label is "X0Y0";
42    attribute rloc of five_LUT          : label is "X0Y0";
43
44    attribute rloc of eight_LUT         : label is "X1Y0";
45    attribute rloc of nine_LUT          : label is "X1Y0";
46    attribute rloc of eleven_LUT        : label is "X1Y0";
47    attribute rloc of twelve_LUT        : label is "X1Y0";
48
49    attribute rloc of fourteen_LUT      : label is "X1Y1";
50    attribute rloc of seventeen_LUT     : label is "X1Y1";
51    attribute rloc of eighteen_LUT      : label is "X1Y1";
52    attribute rloc of twenty_LUT        : label is "X1Y1";
53
54    attribute rloc of csc0_c            : label is "X0Y2";
55    attribute rloc of csc1_c            : label is "X0Y2";
56    attribute rloc of csc2_c            : label is "X0Y2";
57    attribute rloc of rh_LUT            : label is "X0Y2";
58
59    attribute rloc of ri_LUT            : label is "X1Y2";
60    attribute rloc of re_LUT            : label is "X1Y2";
61    attribute rloc of header_flit_c     : label is "X1Y2";
62    attribute rloc of control_flit_c    : label is "X1Y2";
63
64    attribute rloc of addr_flit_c       : label is "X2Y2";
65    attribute rloc of data_flit_c       : label is "X2Y2";
66    attribute rloc of sync_ack_c        : label is "X2Y2";
67
68
69
70  begin
71
72  --  # EQN file for model async_transmitter_lo
73  --  # Generated by petrify 4.2 (compiled 15-Oct-03 at 3:06 PM)
74  --  # Outputs between brackets "[out]" indicate a feedback to input "out"
75  --  # Estimated area = 83.00
76  --
77  --  INORDER = sync_req ack header_flit rh control_flit ri addr_flit data_flit
        sync_ack re csc0 csc1 csc2;
78  --  OUTORDER = [header_flit] [rh] [control_flit] [ri] [addr_flit] [data_flit]
        [sync_ack] [re] [csc0] [csc1] [csc2];
79  --  [0] = csc0' sync_req csc2;
80  --  [1] = ack' csc0 header_flit;
81  --  [header_flit] = [1]' ([0] + header_flit) + header_flit [0];      #
      mappable onto gC
82  --  [rh] = csc0' header_flit;
83  --  [4] = csc0 header_flit' csc2;
84  --  [5] = ack' csc0' control_flit;
85  --  [control_flit] = [5]' ([4] + control_flit) + control_flit [4];     #
      mappable onto gC
```

```
86  --  [ri] = csc0' addr_flit + csc0 control_flit;
87  --  [8] = control_flit' csc1 csc2';
88  --  [9] = ack' csc0 csc1';
89  --  [addr_flit] = [9]' ([8] + addr_flit) + addr_flit [8];      # mappable
        onto gC
90  --  [11] = csc0 addr_flit' csc1';
91  --  [12] = ack' csc0' csc1';
92  --  [data_flit] = [12]' ([11] + data_flit) + data_flit [11];     # mappable
        onto gC
93  --  [14] = csc0' csc1' data_flit';
94  --  [sync_ack] = csc1' ([14] + sync_ack) + sync_ack [14];      # mappable
        onto gC
95  --  [re] = csc0 data_flit;
96  --  [17] = ack (rh + addr_flit);
97  --  [18] = ack addr_flit' csc2';
98  --  [csc0] = [18]' ([17] + csc0) + csc0 [17];      # mappable onto gC
99  --  [20] = csc0 addr_flit;
100 --  [csc1] = [20]' (csc2 + csc1) + csc1 csc2;      # mappable onto gC
101 --  [csc2] = sync_req' (control_flit' + csc2) + control_flit' csc2;      #
        mappable onto gC
102 --
103 --  # Set/reset pins: reset(header_flit) reset(control_flit) reset(addr_flit)
        reset(data_flit) reset(csc0)
104
105 --  [0] = csc0' sync_req csc2;
106
107   zero_LUT : LUT3
108   generic map (
109     INIT => X"08")
110
111   port map (
112     O => zero,
113     I0 => csc2,
114     I1 => sync_req,
115     I2 => csc0
116   );
117
118 --  [1] = ack' csc0 header_flit;
119
120   one_LUT : LUT3
121   generic map (
122     INIT => X"08")
123
124   port map (
125     O => one,
126     I0 => header_flit,
127     I1 => csc0,
128     I2 => ack
129   );
130
131 --  [rh] = csc0' header_flit;
132
133   rh_LUT: LUT2
134   generic map (
135     INIT => X"2")
136
137   port map (
138     O  => rh,
139     I0 => header_flit,
140     I1 => csc0
141   );
142
143 --  [4] = csc0 header_flit' csc2;
144
145   four_LUT : LUT3
```

```
146     generic map (
147       INIT => X"20")
148
149     port map (
150       O => four,
151       I0 => csc2,
152       I1 => header_flit,
153       I2 => csc0
154     );
155
156   --  [5] = ack' csc0' control_flit;
157
158     five_LUT : LUT3
159     generic map (
160       INIT => X"02")
161
162     port map (
163       O => five,
164       I0 => control_flit,
165       I1 => csc0,
166       I2 => ack
167     );
168
169   --  [ri] = csc0' addr_flit + csc0 control_flit;
170
171     ri_LUT : LUT4
172     generic map (
173       INIT => X"88f8")
174
175     port map (
176       O  => ri,
177       I0 => control_flit,
178       I1 => csc0,
179       I2 => addr_flit,
180       I3 => csc0
181     );
182
183   --  [8] = control_flit' csc1 csc2';
184
185     eight_LUT : LUT3
186     generic map (
187       INIT => X"04")
188
189     port map (
190       O => eight,
191       I0 => csc2,
192       I1 => csc1,
193       I2 => control_flit
194     );
195
196   --  [9] = ack' csc0 csc1';
197
198     nine_LUT : LUT3
199     generic map (
200       INIT => X"04")
201
202     port map (
203       O => nine,
204       I0 => csc1,
205       I1 => csc0,
206       I2 => ack
207     );
208
209   --  [11] = csc0 addr_flit' csc1';
210
```

```
211   eleven_LUT : LUT3
212   generic map (
213     INIT => X"10")
214
215   port map (
216     O => eleven,
217     I0 => csc1,
218     I1 => addr_flit,
219     I2 => csc0
220   );
221
222 --  [12] = ack' csc0' csc1';
223
224   twelve_LUT : LUT3
225   generic map (
226     INIT => X"01")
227
228   port map (
229     O => twelve,
230     I0 => csc1,
231     I1 => csc0,
232     I2 => ack
233   );
234
235 --  [14] = csc0' csc1' data_flit';
236
237   fourteen_LUT : LUT3
238   generic map (
239     INIT => X"01")
240
241   port map (
242     O => fourteen,
243     I0 => data_flit,
244     I1 => csc1,
245     I2 => csc0
246   );
247
248 --  [re] = csc0 data_flit;
249
250   re_LUT: LUT2
251   generic map (
252     INIT => X"8")
253
254   port map (
255     O  => re,
256     I0 => data_flit,
257     I1 => csc0
258   );
259
260 --  [17] = ack (rh + addr_flit);
261
262   seventeen_LUT : LUT3
263   generic map (
264     INIT => X"e0")
265
266   port map (
267     O => seventeen,
268     I0 => addr_flit,
269     I1 => rh,
270     I2 => ack
271   );
272
273 --  [18] = ack addr_flit' csc2';
274
275   eighteen_LUT : LUT3
```

```
276     generic map (
277       INIT => X"10")
278
279     port map (
280       O => eighteen,
281       I0 => csc2,
282       I1 => addr_flit,
283       I2 => ack
284     );
285
286   --  [20] = csc0 addr_flit;
287
288     twenty_LUT: LUT2
289     generic map (
290       INIT => X"8")
291
292     port map (
293       O  => twenty,
294       I0 => addr_flit,
295       I1 => csc0
296     );
297
298
299     -- C-elements
300
301   --  [header_flit] = [1]' ([0] + header_flit) + header_flit [0];      #
        mappable onto gC
302
303     -- C-element with inverted i1 input
304     header_flit_c: lut4_l
305     generic map (
306       init => "10110010" & x"00"
307     )
308     port map (
309       i0 => zero,
310       i1 => one,
311       i2 => header_flit,
312       i3 => reset,
313       lo => header_flit
314     );
315
316   --  [control_flit] = [5]' ([4] + control_flit) + control_flit [4];     #
        mappable onto gC
317
318     -- C-element with inverted i1 input
319     control_flit_c: lut4_l
320     generic map (
321       init => "10110010" & x"00"
322     )
323     port map (
324       i0 => four,
325       i1 => five,
326       i2 => control_flit,
327       i3 => reset,
328       lo => control_flit
329     );
330
331   --  [addr_flit] = [9]' ([8] + addr_flit) + addr_flit [8];      # mappable
        onto gC
332
333     -- C-element with inverted i1 input
334     addr_flit_c: lut4_l
335     generic map (
336       init => "10110010" & x"00"
337     )
```

```
338     port map (
339       i0 => eight,
340       i1 => nine,
341       i2 => addr_flit,
342       i3 => reset,
343       lo => addr_flit
344     );
345
346   --  [data_flit] = [12]' ([11] + data_flit) + data_flit [11];      # mappable
          onto gC
347
348     -- C-element with inverted i1 input
349     data_flit_c: lut4_l
350     generic map (
351       init => "10110010" & x"00"
352     )
353     port map (
354       i0 => eleven,
355       i1 => twelve,
356       i2 => data_flit,
357       i3 => reset,
358       lo => data_flit
359     );
360
361   --  [sync_ack] = csc1' ([14] + sync_ack) + sync_ack [14];       # mappable
          onto gC
362
363     -- C-element with inverted i1 input
364     sync_ack_c: lut4_l
365     generic map (
366       init => "10110010" & x"00"
367     )
368     port map (
369       i0 => fourteen,
370       i1 => csc1,
371       i2 => sync_ack,
372       i3 => reset,
373       lo => sync_ack
374     );
375
376   --  [csc0] = [18]' ([17] + csc0) + csc0 [17];       # mappable onto gC
377
378     -- C-element with inverted i1 input
379     csc0_c: lut4_l
380     generic map (
381       init => "10110010" & x"00"
382     )
383     port map (
384       i0 => seventeen,
385       i1 => eighteen,
386       i2 => csc0,
387       i3 => reset,
388       lo => csc0
389     );
390
391   --  [csc1] = [20]' (csc2 + csc1) + csc1 csc2;       # mappable onto gC
392
393     -- C-element with inverted i1 input
394     csc1_c: lut4_l
395     generic map (
396       init => "10110010" & x"ff"  -- Set to 1 to avoid glitch during reset.
397     )
398     port map (
399       i0 => csc2,
400       i1 => twenty,
```

```
401       i2 => csc1,
402       i3 => reset,
403       lo => csc1
404     );
405
406   --  [csc2] = sync_req' (control_flit' + csc2) + control_flit' csc2;      #
          mappable onto gC
407
408     -- C-element with inverted i0 and i1 input
409     csc2_c: lut4_1
410     generic map (
411       init => "01110001" & x"ff"  -- Set to 1 to avoid glitch during reset.
412     )
413     port map (
414       i0 => control_flit,
415       i1 => sync_req,
416       i2 => csc2,
417       i3 => reset,
418       lo => csc2
419     );
420
421     -- Assign in/outputs
422     sync_req          <= sync_req_in;
423     sync_ack_in       <= sync_ack;
424     rh_out            <= rh;
425     ri_out            <= ri;
426     re_out            <= re;
427     ack               <= ack_out;
428     header_flit_out   <= header_flit;
429     control_flit_out  <= control_flit;
430     addr_flit_out     <= addr_flit;
431     data_flit_out     <= data_flit;
432
433
434   end Behavioral;
```

A.5.3.9 async_receiver.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   entity async_receiver is
8     port(
9       reset             : in  std_logic;
10      rh_in             : in  std_logic;
11      ri_in             : in  std_logic;
12      re_in             : in  std_logic;
13      ack_in            : out std_logic;
14      data_in           : in  flit_data;
15      sync_req_out      : out std_logic;
16      sync_ack_out      : in  std_logic;
17      header_flit_out   : out flit_data;
18      control_flit_out  : out flit_data;
19      ad_flit0_out      : out flit_data;
20      ad_flit1_out      : out flit_data
21    );
22
23  end async_receiver;
24
25  architecture Behavioral of async_receiver is
```

```
26
27    component async_receiver_hs_ctrl is
28    port(
29      reset             : in  std_logic;
30      sync_req_out      : out std_logic;
31      sync_ack_out      : in  std_logic;
32      rh_in             : in  std_logic;
33      ri_in             : in  std_logic;
34      re_in             : in  std_logic;
35      ack_in            : out std_logic;
36      header_latch_out  : out std_logic;
37      control_latch_out : out std_logic;
38      ad_latch0_out     : out std_logic;
39      ad_latch1_out     : out std_logic
40    );
41    end component;
42
43    signal header_latch_en, control_latch_en, ad_latch0_en, ad_latch1_en :
          std_logic;
44
45  begin
46
47    as_hs_ctrl : async_receiver_hs_ctrl
48    port map(
49      reset             => reset,
50      sync_req_out      => sync_req_out,
51      sync_ack_out      => sync_ack_out,
52      rh_in             => rh_in,
53      ri_in             => ri_in,
54      re_in             => re_in,
55      ack_in            => ack_in,
56      header_latch_out  => header_latch_en,
57      control_latch_out => control_latch_en,
58      ad_latch0_out     => ad_latch0_en,
59      ad_latch1_out     => ad_latch1_en
60    );
61
62    header_latch : process(header_latch_en, data_in)
63    begin
64      if header_latch_en = '1' then
65        header_flit_out <= data_in;
66      end if;
67    end process;
68
69    control_latch : process(control_latch_en, data_in)
70    begin
71      if control_latch_en = '1' then
72        control_flit_out <= data_in;
73      end if;
74    end process;
75
76    ad_latch0 : process(ad_latch0_en, data_in)
77    begin
78      if ad_latch0_en = '1' then
79        ad_flit0_out <= data_in;
80      end if;
81    end process;
82
83    ad_latch1 : process(ad_latch1_en, data_in)
84    begin
85      if ad_latch1_en = '1' then
86        ad_flit1_out <= data_in;
87      end if;
88    end process;
89
```

```
90  end Behavioral;
```

A.5.3.10 async_receiver_hs_ctrl.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   library UNISIM;
7   use UNISIM.VComponents.lut4_l;
8   use UNISIM.VComponents.lut6;
9   use UNISIM.VComponents.lut5;
10  use UNISIM.VComponents.lut4;
11  use UNISIM.VComponents.lut3;
12  use UNISIM.VComponents.lut2;
13
14  entity async_receiver_hs_ctrl is
15    port(
16      reset             : in  std_logic;
17      sync_req_out      : out std_logic;
18      sync_ack_out      : in  std_logic;
19      rh_in             : in  std_logic;
20      ri_in             : in  std_logic;
21      re_in             : in  std_logic;
22      ack_in            : out std_logic;
23      header_latch_out  : out std_logic;
24      control_latch_out : out std_logic;
25      ad_latch0_out     : out std_logic;
26      ad_latch1_out     : out std_logic
27    );
28  end async_receiver_hs_ctrl;
29
30  architecture Behavioral of async_receiver_hs_ctrl is
31
32    signal rh,ri,re,sync_ack,header_latch,ack,control_latch,ad_latch0,ad_latch1
          ,sync_req,csc0,csc1,
33          zero, one, two, four, five, six, eight, nine, eleven, twelve :
                std_logic;
34
35    attribute keep : string;
36    attribute keep of rh,ri,re,sync_ack,header_latch,ack,control_latch,
          ad_latch0,ad_latch1,sync_req,csc0,csc1,
37              zero, one, two, four, five, six, eight, nine, eleven, twelve :
                  signal is "true";
38
39    attribute rloc : string;
40    attribute rloc of zero_LUT          : label is "X0Y0";
41    attribute rloc of one_LUT           : label is "X0Y0";
42    attribute rloc of two_LUT           : label is "X0Y0";
43    attribute rloc of four_LUT          : label is "X1Y0";
44    attribute rloc of five_LUT          : label is "X1Y0";
45    attribute rloc of six_LUT           : label is "X1Y0";
46    attribute rloc of eight_LUT         : label is "X1Y0";
47    attribute rloc of nine_LUT          : label is "X0Y1";
48    attribute rloc of eleven_LUT        : label is "X0Y1";
49    attribute rloc of twelve_LUT        : label is "X0Y1";
50    attribute rloc of sync_req_c        : label is "X0Y1";
51    attribute rloc of header_latch_c    : label is "X1Y1";
52    attribute rloc of ack_c             : label is "X1Y1";
53    attribute rloc of control_latch_c   : label is "X1Y1";
54    attribute rloc of ad_latch0_c       : label is "X1Y1";
55    attribute rloc of ad_latch1_c       : label is "X0Y2";
```

```
56    attribute rloc of csc0_c            : label is "X0Y2";
57    attribute rloc of csc1_c            : label is "X0Y2";
58
59
60
61  begin
62
63  --# EQN file for model async_receive
64  --# Generated by petrify 4.2 (compiled 15-Oct-03 at 3:06 PM)
65  --# Outputs between brackets "[out]" indicate a feedback to input "out"
66  --# Estimated area = 75.00
67  --
68  --INORDER = rh ri re sync_ack header_latch ack control_latch ad_latch0
        ad_latch1 sync_req csc0 csc1;
69  --OUTORDER = [header_latch] [ack] [control_latch] [ad_latch0] [ad_latch1] [
        sync_req] [csc0] [csc1];
70  --[0] = sync_ack' rh csc0;
71  --[1] = ri csc0 csc1 + csc1' (ad_latch1 + ad_latch0) + header_latch;
72  --[2] = ad_latch0' control_latch' header_latch' ad_latch1';
73  --[ack] = [2]' ([1] + ack) + ack [1];         # mappable onto gC
74  --[4] = ri csc1';
75  --[5] = csc0' (ri + re);
76  --[6] = ad_latch0 re' ri';
77  --[ad_latch0] = [6]' ([5] + ad_latch0) + ad_latch0 [5];      # mappable onto
        gC
78  --[8] = re csc0;
79  --[9] = ack' re' csc0' csc1';
80  --[sync_req] = csc0' ([9] + sync_req) + sync_req [9];        # mappable onto
        gC
81  --[11] = ad_latch0 ri + sync_ack;
82  --[12] = re' ad_latch1 + control_latch ri';
83  --[csc0] = [12]' ([11] + csc0) + csc0 [11];          # mappable onto gC
84  --[csc1] = re' (control_latch + csc1) + control_latch csc1;          #
        mappable onto gC
85  --[header_latch] = rh ([0] + header_latch) + header_latch [0];       #
        mappable onto gC
86  --[control_latch] = csc0 ([4] + control_latch) + control_latch [4];
        # mappable onto gC
87  --[ad_latch1] = csc0 ([8] + ad_latch1) + ad_latch1 [8];      # mappable onto
        gC
88  --
89  --# Set/reset pins: reset(ad_latch0) set(csc0) reset(csc1) reset(
        control_latch) reset(ad_latch1)
90
91   --zero: [0] = sync_ack' rh csc0;
92
93    zero_LUT : LUT3
94    generic map (
95      INIT => X"08")
96
97    port map (
98      O => zero,
99      I0 => csc0,
100     I1 => rh,
101     I2 => sync_ack
102   );
103
104 --[1] = ri csc0 csc1 + csc1' (ad_latch1 + ad_latch0) + header_latch;
105
106   one_LUT : LUT6
107   generic map (
108     INIT => X"fffeaafeaafeaafe")
109
110   port map (
111     O  => one,
```

```
112       I0 => header_latch,
113       I1 => ad_latch0,
114       I2 => ad_latch1,
115       I3 => csc1,
116       I4 => csc0,
117       I5 => ri
118     );
119
120   --[2] = ad_latch0' control_latch' header_latch' ad_latch1';
121
122     two_LUT : LUT4
123     generic map (
124       INIT => X"0001")
125
126     port map (
127       O  => two,
128       I0 => ad_latch1,
129       I1 => header_latch,
130       I2 => control_latch,
131       I3 => ad_latch0
132     );
133
134   --[4] = ri csc1';
135
136     four_LUT: LUT2
137     generic map (
138       INIT => X"4")
139
140     port map (
141       O  => four,
142       I0 => csc1,
143       I1 => ri
144     );
145
146   --[5] = csc0' (ri + re);
147
148     five_LUT : LUT3
149     generic map (
150       INIT => X"0e")
151
152     port map (
153       O  => five,
154       I0 => re,
155       I1 => ri,
156       I2 => csc0
157     );
158
159   --[6] = ad_latch0 re' ri';
160
161     six_LUT : LUT3
162     generic map (
163       INIT => X"10")
164
165     port map (
166       O  => six,
167       I0 => ri,
168       I1 => re,
169       I2 => ad_latch0
170     );
171
172   --[8] = re csc0;
173
174     eight_LUT : LUT2
175     generic map (
176       INIT => X"8")
```

```
177
178   port map (
179     O  => eight,
180     I0 => csc0,
181     I1 => re
182   );
183
184 --[9] = ack' re' csc0' csc1';
185
186   nine_LUT : LUT4
187   generic map (
188     INIT => X"0001")
189
190   port map (
191     O  => nine,
192     I0 => ack,
193     I1 => re,
194     I2 => csc0,
195     I3 => csc1
196   );
197
198 --[11] = ad_latch0 ri + sync_ack;
199
200   eleven_LUT : LUT3
201   generic map (
202     INIT => X"ea")
203
204   port map (
205     O  => eleven,
206     I0 => sync_ack,
207     I1 => ri,
208     I2 => ad_latch0
209   );
210
211 --[12] = re' ad_latch1 + control_latch ri';
212
213   twelve_LUT : LUT4
214   generic map (
215     INIT => X"44f4")
216
217   port map (
218     O  => twelve,
219     I0 => ri,
220     I1 => control_latch,
221     I2 => ad_latch1,
222     I3 => re
223   );
224
225   -- C-element with inverted i1 input
226   ack_c: lut4_l
227   generic map (
228     init => "10110010" & x"00"
229   )
230   port map (
231     i0 => one,
232     i1 => two,
233     i2 => ack,
234     i3 => reset,
235     lo => ack
236   );
237
238   -- C-element with inverted i1 input
239   ad_latch0_c: lut4_l
240   generic map (
241     init => "10110010" & x"00"
```

```
242     )
243     port map (
244       i0 => five,
245       i1 => six,
246       i2 => ad_latch0,
247       i3 => reset,
248       lo => ad_latch0
249     );
250
251
252     -- C-element with inverted i1 input
253     sync_req_c: lut4_l
254     generic map (
255       init => "10110010" & x"00"
256     )
257     port map (
258       i0 => nine,
259       i1 => csc0,
260       i2 => sync_req,
261       i3 => reset,
262       lo => sync_req
263     );
264
265     -- C-element with inverted i1 input
266     csc0_c: lut4_l
267     generic map (
268       init => "10110010" & x"11" -- initialize to 1
269     )
270     port map (
271       i0 => eleven,
272       i1 => twelve,
273       i2 => csc0,
274       i3 => reset,
275       lo => csc0
276     );
277
278     -- C-element with inverted i1 input
279     csc1_c: lut4_l
280     generic map (
281       init => "10110010" & x"00"
282     )
283     port map (
284       i0 => control_latch,
285       i1 => re,
286       i2 => csc1,
287       i3 => reset,
288       lo => csc1
289     );
290
291     --C-element
292     header_latch_c: lut4_l
293     generic map (
294       init => "11101000" & x"00"
295     )
296     port map (
297       i0 => rh,
298       i1 => zero,
299       i2 => header_latch,
300       i3 => reset,
301       lo => header_latch
302     );
303
304     --C-element
305     control_latch_c: lut4_l
306     generic map (
```

```
307       init => "11101000" & x"00"
308     )
309     port map (
310       i0 => csc0,
311       i1 => four,
312       i2 => control_latch,
313       i3 => reset,
314       lo => control_latch
315     );
316
317     --C-element
318     ad_latch1_c: lut4_l
319     generic map (
320       init => "11101000" & x"00"
321     )
322     port map (
323       i0 => csc0,
324       i1 => eight,
325       i2 => ad_latch1,
326       i3 => reset,
327       lo => ad_latch1
328     );
329
330     -- Assign in/outputs
331     sync_req_out      <= sync_req;
332     sync_ack          <= sync_ack_out;
333     rh                <= rh_in;
334     ri                <= ri_in;
335     re                <= re_in;
336     ack_in            <= ack;
337     header_latch_out  <= header_latch;
338     control_latch_out <= control_latch;
339     ad_latch0_out     <= ad_latch0;
340     ad_latch1_out     <= ad_latch1;
341
342   end Behavioral;
```

A.5.4 Traffic Generator

A.5.4.1 traffic_source.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6   use work.source_rom_data.all;
7
8   entity traffic_source is
9     generic(
10      ROM : rom_type := ROM_ZERO
11    );
12    port(
13      reset     : in  std_logic;
14      rh_out    : out STD_LOGIC;
15      ri_out    : out STD_LOGIC;
16      re_out    : out STD_LOGIC;
17      ack_out   : in  STD_LOGIC;
18      data_out  : out flit_data
19    );
20  end traffic_source;
```

```
21
22  architecture Behavioral of traffic_source is
23
24    component as_bd_4p_delay is
25      generic(
26        size : natural range 1 to 30 := 10  -- Delay size
27      );
28      port (
29        d : in  std_logic;     -- Data in
30        z : out std_logic      -- Data out
31      );
32    end component;
33
34    signal req, req_delayed, rom_clk, ack, reset_delayed : std_logic;
35    signal req_type : std_logic_vector(1 downto 0);
36
37    signal count : unsigned(5 downto 0);
38
39    signal rom_value : std_logic_vector(FLIT_SIZE+1 downto 0);
40
41    attribute keep : string;
42    attribute keep of req, req_delayed, rom_clk, ack, reset_delayed, req_type,
          count, rom_value  : signal is "true";
43
44  begin
45
46    req <= (not reset_delayed) or (not ack);
47    rom_clk <= not((not reset) or (not req));
48    ack <= ack_out;
49
50    req_delay : as_bd_4p_delay
51    generic map(
52      size => 10
53    )
54    port map(
55      d => req,
56      z => req_delayed
57    );
58
59    reset_delay : as_bd_4p_delay
60    generic map(
61      size => 10
62    )
63    port map(
64      d => reset,
65      z => reset_delayed
66    );
67
68    req_control : process(req_type, req_delayed)
69    begin
70      if req_type  = "01" then      -- rh
71          rh_out <= req_delayed;
72          ri_out <= '0';
73          re_out <= '0';
74      elsif req_type = "10" then    -- ri
75          rh_out <= '0';
76          ri_out <= req_delayed;
77          re_out <= '0';
78      elsif req_type = "11" then    -- re
79          rh_out <= '0';
80          ri_out <= '0';
81          re_out <= req_delayed;
82      else
83          rh_out <= '0';
84          ri_out <= '0';
```

```
85          re_out <= '0';
86        end if;
87      end process;
88
89      counter : process(reset, ack_out)
90      begin
91        if reset = '0' then
92          count <= "000000";
93        elsif rising_edge(ack_out) then
94          if count < 2 then
95            count <= count + 1;
96          else
97            count <= "000000";
98          end if;
99        end if;
100     end process;
101
102     --Inferred ROM  (Will be inferred as block ram. But no reset, it'll mess it
            up!)
103     rom_block : process(rom_clk)
104     begin
105       if rising_edge(rom_clk) then
106         rom_value <= ROM(conv_integer(count));
107       end if;
108     end process;
109
110     data_out <= rom_value(FLIT_SIZE-1 downto 0);
111     req_type <= rom_value(FLIT_SIZE+1 downto FLIT_SIZE);    --two MSBs
112
113   end Behavioral;
```

A.5.4.2 traffic_sink.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7
8   entity traffic_sink is
9     port(
10      reset     : in  std_logic;
11      rh_in     : in  STD_LOGIC;
12      ri_in     : in  STD_LOGIC;
13      re_in     : in  STD_LOGIC;
14      ack_in    : out STD_LOGIC;
15      alive     : out std_logic;
16
17      -- ILA Signals --
18      ILA_clk   : out std_logic
19    );
20  end traffic_sink;
21
22  architecture Behavioral of traffic_sink is
23
24    ------------------------------------------------------------------
25    --
26    --  ILA core component declaration
27    --
28    ------------------------------------------------------------------
29    component ila
30      port
```

```
31      (
32        control     : in    std_logic_vector(35 downto 0);
33        clk         : in    std_logic;
34        trig0       : in    flit_data
35      );
36    end component;
37
38    component as_bd_4p_delay is
39      generic(
40        size : natural range 1 to 30 := 10  -- Delay size
41      );
42      port (
43        d : in  std_logic;     -- Data in
44        z : out std_logic      -- Data out
45      );
46    end component;
47
48    signal req_in, req_delayed : std_logic;
49    signal data : flit_data;
50
51    signal count : unsigned(24 downto 0);
52
53
54  begin
55
56    ------------------------------------------------------------------
57    --
58    --  ILA core instance
59    --  Note: If hierarchy is kept, it is not possible to instantiate the ILA
          core in a sub-entity
60    ------------------------------------------------------------------
61  --  i_ila : ila
62  --    port map
63  --    (
64  --      control   => ILA_control,
65  --      clk       => req_in,
66  --      trig0     => data_in
67  --    );
68
69    ILA_clk <= req_in;
70
71    req_in <= rh_in or ri_in or re_in;
72    ack_in <= req_delayed;
73
74
75    --Delay must be quite large before it works in chipscope. 4 doesn't work -
          10 does!
76    --It might not be chipscope that is the problem, it should work with
          frequencies up to 500Mhz.
77    ack_delay : as_bd_4p_delay
78    generic map(
79      size => 10
80    )
81    port map(
82      d => req_in,
83      z => req_delayed
84    );
85
86  end Behavioral;
```

A.5.4.3 source_rom_data.vhd

```
1   -- Traffic Source ROM initialization values
```

```
2
3  library IEEE;
4  use IEEE.std_logic_1164.all;
5  use work.types.all;
6
7  package source_rom_data is
8
9    constant ROM_ZERO : rom_type := (others => "00"&FLIT_ZERO);
10
11   constant north_source_rom_data : rom_type := (
12     --east  (0100010101010000) -> (0001010101000001) "1541"
13     0       => "01" & x"4550",
14     1       => "10" & x"AAA0",
15     2       => "11" & x"5550",
16     --south (1000101010100000) -> (0010101010000010) "2A82"
17     3       => "01" & x"8AA0",
18     4       => "10" & x"5550",
19     5       => "11" & x"AAA0",
20     --west  (1100010101010000) -> (0001010101000011) "1543"
21     6       => "01" & x"c550",
22     7       => "10" & x"AAA0",
23     8       => "11" & x"5550",
24     --local (0000101010100000) -> (0010101010000000) "2A80"
25     9       => "01" & x"0AA0",
26     10      => "10" & x"5550",
27     11      => "11" & x"AAA0",
28     others  => "00" & x"0000"
29   );
30
31   constant east_source_rom_data : rom_type := (
32     --north (0000010101010001) -> (0001010101000100) "1544"
33     0       => "01" & x"0551",
34     1       => "10" & x"AAA1",
35     2       => "11" & x"5551",
36     --south (1000101010100001) -> (0010101010000110) "2A86"
37     3       => "01" & x"8AA1",
38     4       => "10" & x"5551",
39     5       => "11" & x"AAA1",
40     --west  (1100010101010001) -> (0001010101000111) "1547"
41     6       => "01" & x"c551",
42     7       => "10" & x"AAA1",
43     8       => "11" & x"5551",
44     --local (0100101010100001) -> (0010101010000101) "2A85"
45     9       => "01" & x"4AA1",
46     10      => "10" & x"5551",
47     11      => "11" & x"AAA1",
48     others  => "00" & x"0000"
49   );
50
51   constant south_source_rom_data : rom_type := (
52     --north  (0000010101010010) -> (0001010101001000) "1548"
53     0       => "01" & x"0552",
54     1       => "10" & x"AAA2",
55     2       => "11" & x"5552",
56     --east   (0100101010100010) -> (0010101010001001) "2A89"
57     3       => "01" & x"4AA2",
58     4       => "10" & x"5552",
59     5       => "11" & x"AAA2",
60     --west   (1100010101010010) -> (0001010101001011) "154b"
61     6       => "01" & x"c552",
62     7       => "10" & x"AAA2",
63     8       => "11" & x"5552",
64     --local  (1000101010100010) -> (0010101010001010) "2A8A"
65     9       => "01" & x"8AA2",
66     10      => "10" & x"5552",
```

```
67      11      => "11" & x"AAA2",
68      others  => "00" & x"0000"
69    );
70
71    constant west_source_rom_data : rom_type  := (
72      --north  (0000010101010011) -> (0001010101001100) "154c"
73      0       => "01" & x"0553",
74      1       => "10" & x"AAA3",
75      2       => "11" & x"5553",
76      --east   (0100101010100011) -> (0010101010001101) "2A8d"
77      3       => "01" & x"4AA3",
78      4       => "10" & x"5553",
79      5       => "11" & x"AAA3",
80      --south  (1000010101010011) -> (0001010101001110) "154E"
81      6       => "01" & x"8553",
82      7       => "10" & x"AAA3",
83      8       => "11" & x"5553",
84      --local  (1100101010100011) -> (0010101010001111) "2A8F"
85      9       => "01" & x"cAA3",
86      10      => "10" & x"AAA3",
87      11      => "11" & x"5553",
88      others  => "00" & x"0000"
89    );
90
91    constant local_source_rom_data : rom_type := (
92      --north  (0000010101010100) -> (0001010101010000) "1550"
93      0       => "01" & x"0554",
94      1       => "10" & x"AAA4",
95      2       => "11" & x"5554",
96      --east   (0100101010100100) -> (0010101010010001) "2A91"
97      3       => "01" & x"4AA4",
98      4       => "10" & x"5554",
99      5       => "11" & x"AAA4",
100     --south  (1000010101010100) -> (0001010101010010) "1552"
101     6       => "01" & x"8554",
102     7       => "10" & x"AAA4",
103     8       => "11" & x"5554",
104     --west   (1100101010100100) -> (0010101010010011) "2A93"
105     9       => "01" & x"cAA4",
106     10      => "10" & x"5554",
107     11      => "11" & x"AAA4",
108     others  => "00" & x"0000"
109   );
110
111 end source_rom_data;
112
113 package body source_rom_data is
114
115 end source_rom_data;
```

A.5.5 MPSoc

A.5.5.1 MPSoC_noc.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6  use work.route_lookup_tables.all;
7
```

```
8  ---- Uncomment the following library declaration if instantiating
9  ---- any Xilinx primitives in this code.
10 --library UNISIM;
11 --use UNISIM.VComponents.bufg;
12
13 entity MPSoC_noc is
14   port(
15     clk0_i  : in  std_logic;
16     clk1_i  : in  std_logic;
17     reset_i : in  std_logic;
18     tx      : out std_logic;
19     rx      : in  std_logic;
20     running : out std_logic
21   );
22 end MPSoC_noc;
23
24 architecture struct of MPSoC_noc is
25
26   component or1200_ocp is
27     port (
28       clk_i            : in  std_logic;       -- Clock
29       rst_i            : in  std_logic;       -- Reset
30
31       -- OCP Master interface signals
32       ocp_MCmd_o       : out MCmdEncoding;    -- OCP master command
33       ocp_Maddr_o      : out std_logic_vector(addr_width-1 downto 0);
34                                               -- OCP master address
35       ocp_MData_o      : out std_logic_vector(data_width-1 downto 0);
36                                               -- OCP master data
37       ocp_MByteEn_o    : out std_logic_vector(3 downto 0);  -- OCP master
             byte enable
38       ocp_SCmdAccept_i : in  std_logic;       -- OCP slave command accept
39       ocp_SResp_i      : in  SRespEncoding;   -- OCP slave response
40       ocp_SData_i      : in  std_logic_vector(data_width-1 downto 0)
41     );
42   end component;
43
44   component noc_mesh is
45     port(
46       reset : in std_logic;
47
48       -- router0 ports
49       --input
50       r0_rh_in    : in  std_logic;
51       r0_ri_in    : in  std_logic;
52       r0_re_in    : in  std_logic;
53       r0_ack_in   : out std_logic;
54       r0_data_in  : in  flit_data;
55       --output
56       r0_rh_out   : out std_logic;
57       r0_ri_out   : out std_logic;
58       r0_re_out   : out std_logic;
59       r0_ack_out  : in  std_logic;
60       r0_data_out : out flit_data;
61
62       -- router1 ports
63       --input
64       r1_rh_in    : in  std_logic;
65       r1_ri_in    : in  std_logic;
66       r1_re_in    : in  std_logic;
67       r1_ack_in   : out std_logic;
68       r1_data_in  : in  flit_data;
69       --output
70       r1_rh_out   : out std_logic;
71       r1_ri_out   : out std_logic;
```

```
72        r1_re_out   : out std_logic;
73        r1_ack_out  : in  std_logic;
74        r1_data_out : out flit_data;
75
76        -- router2 ports
77        --input
78        r2_rh_in    : in  std_logic;
79        r2_ri_in    : in  std_logic;
80        r2_re_in    : in  std_logic;
81        r2_ack_in   : out std_logic;
82        r2_data_in  : in  flit_data;
83        --output
84        r2_rh_out   : out std_logic;
85        r2_ri_out   : out std_logic;
86        r2_re_out   : out std_logic;
87        r2_ack_out  : in  std_logic;
88        r2_data_out : out flit_data;
89
90        -- router3 ports
91        --input
92        r3_rh_in    : in  std_logic;
93        r3_ri_in    : in  std_logic;
94        r3_re_in    : in  std_logic;
95        r3_ack_in   : out std_logic;
96        r3_data_in  : in  flit_data;
97        --output
98        r3_rh_out   : out std_logic;
99        r3_ri_out   : out std_logic;
100       r3_re_out   : out std_logic;
101       r3_ack_out  : in  std_logic;
102       r3_data_out : out flit_data;
103
104       -- router4 ports
105       --input
106       r4_rh_in    : in  std_logic;
107       r4_ri_in    : in  std_logic;
108       r4_re_in    : in  std_logic;
109       r4_ack_in   : out std_logic;
110       r4_data_in  : in  flit_data;
111       --output
112       r4_rh_out   : out std_logic;
113       r4_ri_out   : out std_logic;
114       r4_re_out   : out std_logic;
115       r4_ack_out  : in  std_logic;
116       r4_data_out : out flit_data;
117
118       -- router5 ports
119       --input
120       r5_rh_in    : in  std_logic;
121       r5_ri_in    : in  std_logic;
122       r5_re_in    : in  std_logic;
123       r5_ack_in   : out std_logic;
124       r5_data_in  : in  flit_data;
125       --output
126       r5_rh_out   : out std_logic;
127       r5_ri_out   : out std_logic;
128       r5_re_out   : out std_logic;
129       r5_ack_out  : in  std_logic;
130       r5_data_out : out flit_data;
131
132       --input
133       r5_east_rh_in     : in  std_logic;
134       r5_east_ri_in     : in  std_logic;
135       r5_east_re_in     : in  std_logic;
136       r5_east_ack_in    : out std_logic;
```

```
137       r5_east_data_in   : in  flit_data;
138       --output
139       r5_east_rh_out    : out std_logic;
140       r5_east_ri_out    : out std_logic;
141       r5_east_re_out    : out std_logic;
142       r5_east_ack_out   : in  std_logic;
143       r5_east_data_out  : out flit_data;
144
145       -- router6 ports
146       --input
147       r6_rh_in    : in  std_logic;
148       r6_ri_in    : in  std_logic;
149       r6_re_in    : in  std_logic;
150       r6_ack_in   : out std_logic;
151       r6_data_in  : in  flit_data;
152       --output
153       r6_rh_out   : out std_logic;
154       r6_ri_out   : out std_logic;
155       r6_re_out   : out std_logic;
156       r6_ack_out  : in  std_logic;
157       r6_data_out : out flit_data;
158
159       -- router7 ports
160       --input
161       r7_rh_in    : in  std_logic;
162       r7_ri_in    : in  std_logic;
163       r7_re_in    : in  std_logic;
164       r7_ack_in   : out std_logic;
165       r7_data_in  : in  flit_data;
166       --output
167       r7_rh_out   : out std_logic;
168       r7_ri_out   : out std_logic;
169       r7_re_out   : out std_logic;
170       r7_ack_out  : in  std_logic;
171       r7_data_out : out flit_data;
172
173       -- router8 ports
174       --input
175       r8_rh_in    : in  std_logic;
176       r8_ri_in    : in  std_logic;
177       r8_re_in    : in  std_logic;
178       r8_ack_in   : out std_logic;
179       r8_data_in  : in  flit_data;
180       --output
181       r8_rh_out   : out std_logic;
182       r8_ri_out   : out std_logic;
183       r8_re_out   : out std_logic;
184       r8_ack_out  : in  std_logic;
185       r8_data_out : out flit_data
186     );
187     end component;
188
189   component master_na is
190   generic(
191     routing_table : route_lookup_table_type
192   );
193   port (
194     clk_i   : in std_logic;
195     reset_i : in std_logic;
196
197     -- OCP interface
198     ocp_MCmd_i          : in  MCmdEncoding;
199     ocp_Maddr_i         : in  std_logic_vector(addr_width-1 downto 0);
200     ocp_MData_i         : in  std_logic_vector(addr_width-1 downto 0);
201     ocp_MByteEn_i       : in  std_logic_vector(3 downto 0);
```

```
202       ocp_SCmdAccept_o    : out std_logic;
203       ocp_SResp_o         : out SRespEncoding;
204       ocp_SData_o         : out std_logic_vector(addr_width-1 downto 0);
205
206       -- transmit hs channel
207       rh_out    : out std_logic;
208       ri_out    : out std_logic;
209       re_out    : out std_logic;
210       ack_out   : in  std_logic;
211       data_out  : out flit_data;
212
213       -- receive hs channel
214       rh_in   : in  std_logic;
215       ri_in   : in  std_logic;
216       re_in   : in  std_logic;
217       ack_in  : out std_logic;
218       data_in : in  flit_data
219
220     );
221     end component;
222
223     component slave_na is
224     port(
225       clk_i   : in std_logic;
226       reset_i : in std_logic;
227
228       -- OCP interface
229       ocp_MCmd_o          : out MCmdEncoding;
230       ocp_Maddr_o         : out std_logic_vector(addr_width-1 downto 0);
231       ocp_MData_o         : out std_logic_vector(addr_width-1 downto 0);
232       ocp_MByteEn_o       : out std_logic_vector(3 downto 0);
233       ocp_SCmdAccept_i    : in  std_logic;
234       ocp_SResp_i         : in  SRespEncoding;
235       ocp_SData_i         : in  std_logic_vector(addr_width-1 downto 0);
236
237       -- transmit hs channel
238       rh_out    : out std_logic;
239       ri_out    : out std_logic;
240       re_out    : out std_logic;
241       ack_out   : in  std_logic;
242       data_out  : out flit_data;
243
244       -- receive hs channel
245       rh_in   : in  std_logic;
246       ri_in   : in  std_logic;
247       re_in   : in  std_logic;
248       ack_in  : out std_logic;
249       data_in : in  flit_data
250
251     );
252     end component;
253
254     component uart16550_ocp is
255     port (
256       clk_i            : in  std_logic;      -- Clock
257       rst_i            : in  std_logic;      -- Reset
258
259       -- OCP slave interface
260       ocp_MCmd_i       : in  MCmdEncoding;   -- OCP master command
261       ocp_Maddr_i      : in  std_logic_vector(addr_width-1 downto 0);
262                                              -- OCP master address
263       ocp_MData_i      : in  std_logic_vector(data_width-1 downto 0);
264                                              -- OCP master data
265       ocp_MByteEn_i    : in std_logic_vector(3 downto 0); -- OCP master
              byteenable
```

```
266       ocp_SCmdAccept_o : out std_logic;       -- OCP slave command accept
267       ocp_SResp_o      : out SRespEncoding;   -- OCP slave response
268       ocp_SData_o      : out std_logic_vector(data_width-1 downto 0);
269       -- Interrupt
270       int_o            : out std_logic;    -- Interrupt signal
271
272       -- RS232 interface
273       tx               : out std_logic;    -- TX pad
274       rx               : in  std_logic;    -- RX pad
275       rts              : out std_logic;    -- RTS pad
276       cts              : in  std_logic;    -- CTS pad
277       dtr              : out std_logic;    -- DTR pad
278       dsr              : in  std_logic;    -- DSR pad
279       ri               : in  std_logic;    -- RI pad
280       dcd              : in  std_logic);   -- DCD pad
281     end component; -- uart16550_ocp;
282
283     component core_mem_ocp is
284     port (
285       clk_i : in std_logic;                -- Clock
286       rst_i : in std_logic;                -- Reset
287       ocp_MCmd_i : in MCmdEncoding;        -- OCP master command
288       ocp_MAddr_i : in std_logic_vector(addr_width-1 downto 0);
289                                            -- OCP master address
290       ocp_MData_i : in std_logic_vector(data_width-1 downto 0); -- OCP master
              data
291       ocp_MByteEn_i : in std_logic_vector(3 downto 0);  -- OCP Master byte
              enable
292       ocp_SCmdAccept_o : out std_logic;    -- OCP slave command accept
293       ocp_SResp_o : out SRespEncoding;     -- OCP slave response
294       ocp_SData_o : out std_logic_vector(data_width-1 downto 0) -- OCP slave
              data
295       );
296     end component; -- core_mem_ocp;
297
298     component semaphore_ocp is
299     generic (
300       semaphores : integer := 5);        -- log2(Number of semaphores)
301     port (
302       clk_i : in std_logic;                -- Clock
303       rst_i : in std_logic;                -- Reset
304       -- OCP Slave interface
305       ocp_MCmd_i       : in  MCmdEncoding;    -- OCP master command
306       ocp_Maddr_i      : in  std_logic_vector(addr_width-1 downto 0);
307                                               -- OCP master address
308       ocp_MData_i      : in  std_logic_vector(data_width-1 downto 0);
309                                               -- OCP master data
310       ocp_MByteEn_i    : in std_logic_vector(3 downto 0); -- OCP master
              byteenable
311       ocp_SCmdAccept_o : out std_logic;       -- OCP slave command accept
312       ocp_SResp_o      : out SRespEncoding;   -- OCP slave response
313       ocp_SData_o      : out std_logic_vector(data_width-1 downto 0) -- OCP
              slave data
314       );
315     end component;
316
317     component dcm_comp
318       port(
319         clkin_in        : in std_logic;
320         rst_in          : in std_logic;
321         clkdv_out       : out std_logic;
322         clkin_ibufg_out : out std_logic;
323         clk0_out        : out std_logic;
324         locked_out      : out std_logic
325       );
```

```
326   end component;
327
328   component dcm_comp2
329     port(
330       clkin_in        : in std_logic;
331       rst_in          : in std_logic;
332       clkdv_out       : out std_logic;
333       clkin_ibufg_out : out std_logic;
334       clk0_out        : out std_logic;
335       locked_out      : out std_logic
336     );
337   end component;
338
339   component bufg
340     port(
341       o : out std_ulogic;
342       i : in  std_ulogic
343     );
344   end component;
345
346   -- Clock signal
347   signal clk, clk2, dcm_locked2, reset, reset_inv, reset_dcm, dcm_locked, rst
          : std_logic;
348   signal clkcount   : unsigned(23 downto 0) := "000000000000000000000000";
349
350   -- OCP SIGNALS
351
352   -- cpu0 --
353   signal cpu0_MCmd_o       : MCmdEncoding;
354   signal cpu0_Maddr_o      : std_logic_vector(addr_width-1 downto 0);
355   signal cpu0_MData_o      : std_logic_vector(data_width-1 downto 0);
356   signal cpu0_MByteEn_o    : std_logic_vector(3 downto 0);
357   signal cpu0_SCmdAccept_i : std_logic;
358   signal cpu0_SResp_i      : SRespEncoding;
359   signal cpu0_SData_i      : std_logic_vector(data_width-1 downto 0);
360
361   -- cpu1 --
362   signal cpu1_MCmd_o       : MCmdEncoding;
363   signal cpu1_Maddr_o      : std_logic_vector(addr_width-1 downto 0);
364   signal cpu1_MData_o      : std_logic_vector(data_width-1 downto 0);
365   signal cpu1_MByteEn_o    : std_logic_vector(3 downto 0);
366   signal cpu1_SCmdAccept_i : std_logic;
367   signal cpu1_SResp_i      : SRespEncoding;
368   signal cpu1_SData_i      : std_logic_vector(data_width-1 downto 0);
369
370   -- cpu2 --
371   signal cpu2_MCmd_o       : MCmdEncoding;
372   signal cpu2_Maddr_o      : std_logic_vector(addr_width-1 downto 0);
373   signal cpu2_MData_o      : std_logic_vector(data_width-1 downto 0);
374   signal cpu2_MByteEn_o    : std_logic_vector(3 downto 0);
375   signal cpu2_SCmdAccept_i : std_logic;
376   signal cpu2_SResp_i      : SRespEncoding;
377   signal cpu2_SData_i      : std_logic_vector(data_width-1 downto 0);
378
379   -- cpu3 --
380   signal cpu3_MCmd_o       : MCmdEncoding;
381   signal cpu3_Maddr_o      : std_logic_vector(addr_width-1 downto 0);
382   signal cpu3_MData_o      : std_logic_vector(data_width-1 downto 0);
383   signal cpu3_MByteEn_o    : std_logic_vector(3 downto 0);
384   signal cpu3_SCmdAccept_i : std_logic;
385   signal cpu3_SResp_i      : SRespEncoding;
386   signal cpu3_SData_i      : std_logic_vector(data_width-1 downto 0);
387
388   -- cpu4 --
389   signal cpu4_MCmd_o       : MCmdEncoding;
```

```
390     signal cpu4_Maddr_o      : std_logic_vector(addr_width-1 downto 0);
391     signal cpu4_MData_o      : std_logic_vector(data_width-1 downto 0);
392     signal cpu4_MByteEn_o    : std_logic_vector(3 downto 0);
393     signal cpu4_SCmdAccept_i : std_logic;
394     signal cpu4_SResp_i      : SRespEncoding;
395     signal cpu4_SData_i      : std_logic_vector(data_width-1 downto 0);
396
397     -- semaphore --
398     signal semaphore_MCmd_i        : MCmdEncoding;
399     signal semaphore_Maddr_i      : std_logic_vector(addr_width-1 downto 0);
400     signal semaphore_MData_i      : std_logic_vector(data_width-1 downto 0);
401     signal semaphore_MByteEn_i    : std_logic_vector(3 downto 0);
402     signal semaphore_SCmdAccept_o : std_logic;
403     signal semaphore_SResp_o      : SRespEncoding;
404     signal semaphore_SData_o      : std_logic_vector(data_width-1 downto 0);
405
406     -- mem0 --
407     signal mem0_MCmd_i       : MCmdEncoding;
408     signal mem0_Maddr_i      : std_logic_vector(addr_width-1 downto 0);
409     signal mem0_MData_i      : std_logic_vector(data_width-1 downto 0);
410     signal mem0_MByteEn_i    : std_logic_vector(3 downto 0);
411     signal mem0_SCmdAccept_o : std_logic;
412     signal mem0_SResp_o      : SRespEncoding;
413     signal mem0_SData_o      : std_logic_vector(data_width-1 downto 0);
414
415     -- uart --
416     signal uart_MCmd_i       : MCmdEncoding;
417     signal uart_Maddr_i      : std_logic_vector(addr_width-1 downto 0);
418     signal uart_MData_i      : std_logic_vector(data_width-1 downto 0);
419     signal uart_MByteEn_i    : std_logic_vector(3 downto 0);
420     signal uart_SCmdAccept_o : std_logic;
421     signal uart_SResp_o      : SRespEncoding;
422     signal uart_SData_o      : std_logic_vector(data_width-1 downto 0);
423
424     -- HS CHANNELS --
425     signal  cpu0_rh_in,       cpu0_ri_in,       cpu0_re_in,       cpu0_ack_in,
426             cpu0_rh_out,      cpu0_ri_out,      cpu0_re_out,      cpu0_ack_out,
427             cpu1_rh_in,       cpu1_ri_in,       cpu1_re_in,       cpu1_ack_in,
428             cpu1_rh_out,      cpu1_ri_out,      cpu1_re_out,      cpu1_ack_out,
429             cpu2_rh_in,       cpu2_ri_in,       cpu2_re_in,       cpu2_ack_in,
430             cpu2_rh_out,      cpu2_ri_out,      cpu2_re_out,      cpu2_ack_out,
431             cpu3_rh_in,       cpu3_ri_in,       cpu3_re_in,       cpu3_ack_in,
432             cpu3_rh_out,      cpu3_ri_out,      cpu3_re_out,      cpu3_ack_out,
433             cpu4_rh_in,       cpu4_ri_in,       cpu4_re_in,       cpu4_ack_in,
434             cpu4_rh_out,      cpu4_ri_out,      cpu4_re_out,      cpu4_ack_out,
435             semaphore_rh_in,  semaphore_ri_in,  semaphore_re_in,
                    semaphore_ack_in,
436             semaphore_rh_out, semaphore_ri_out, semaphore_re_out,
                    semaphore_ack_out,
437             mem0_rh_in,       mem0_ri_in,       mem0_re_in,       mem0_ack_in,
438             mem0_rh_out,      mem0_ri_out,      mem0_re_out,      mem0_ack_out,
439             uart_rh_in,       uart_ri_in,       uart_re_in,       uart_ack_in,
440             uart_rh_out,      uart_ri_out,      uart_re_out,      uart_ack_out
                      : std_logic;
441
442     signal  cpu0_data_in, cpu0_data_out, cpu1_data_in, cpu1_data_out,
443             cpu2_data_in, cpu2_data_out, cpu3_data_in, cpu3_data_out,
444             cpu4_data_in, cpu4_data_out, mem0_data_in, mem0_data_out,
445             uart_data_in, uart_data_out, semaphore_data_in, semaphore_data_out
                   : flit_data;
446
447   begin
448
449     dcm0 : dcm_comp
450     port map(
```

```
451       clkin_in        => clk0_i,
452       rst_in          => reset_dcm,
453       clkdv_out       => clk,
454       clkin_ibufg_out => open,
455       clk0_out        => open,
456       locked_out      => dcm_locked
457     );
458
459     dcm1 : dcm_comp2
460     port map(
461       clkin_in        => clk1_i,
462       rst_in          => reset_dcm,
463       clkdv_out       => clk2,
464       clkin_ibufg_out => open,
465       clk0_out        => open,
466       locked_out      => dcm_locked2
467     );
468
469     reset_dcm <= not reset_i;
470     reset <= reset_i and dcm_locked and dcm_locked2;
471     reset_inv <= not reset;
472
473     process(clk2)
474     begin
475       if rising_edge(clk2) then
476         clkcount <= clkcount + 1;
477       end if;
478     end process;
479     running <= conv_std_logic_vector(clkcount, 24)(23) and reset;
480
481     cpu0 : or1200_ocp
482       port map(
483         clk_i             => clk2,
484         rst_i             => reset,
485
486         -- OCP Master interface signals
487         ocp_MCmd_o        => cpu0_MCmd_o,
488         ocp_Maddr_o       => cpu0_Maddr_o,
489         ocp_MData_o       => cpu0_MData_o,
490         ocp_MByteEn_o     => cpu0_MByteEn_o,
491         ocp_SCmdAccept_i  => cpu0_SCmdAccept_i,
492         ocp_SResp_i       => cpu0_SResp_i,
493         ocp_SData_i       => cpu0_SData_i
494       );
495
496     cpu1 : or1200_ocp
497       port map(
498         clk_i             => clk2,
499         rst_i             => reset,
500
501         -- OCP Master interface signals
502         ocp_MCmd_o        => cpu1_MCmd_o,
503         ocp_Maddr_o       => cpu1_Maddr_o,
504         ocp_MData_o       => cpu1_MData_o,
505         ocp_MByteEn_o     => cpu1_MByteEn_o,
506         ocp_SCmdAccept_i  => cpu1_SCmdAccept_i,
507         ocp_SResp_i       => cpu1_SResp_i,
508         ocp_SData_i       => cpu1_SData_i
509       );
510
511     cpu2 : or1200_ocp
512       port map(
513         clk_i             => clk,
514         rst_i             => reset,
515
```

```
516         -- OCP Master interface signals
517         ocp_MCmd_o         => cpu2_MCmd_o,
518         ocp_Maddr_o        => cpu2_Maddr_o,
519         ocp_MData_o        => cpu2_MData_o,
520         ocp_MByteEn_o      => cpu2_MByteEn_o,
521         ocp_SCmdAccept_i   => cpu2_SCmdAccept_i,
522         ocp_SResp_i        => cpu2_SResp_i,
523         ocp_SData_i        => cpu2_SData_i
524       );
525
526   --  cpu3 : or1200_ocp
527   --    port map(
528   --      clk_i              => clk,
529   --      rst_i              => reset,
530   --
531   --      -- OCP Master interface signals
532   --      ocp_MCmd_o         => cpu3_MCmd_o,
533   --      ocp_Maddr_o        => cpu3_Maddr_o,
534   --      ocp_MData_o        => cpu3_MData_o,
535   --      ocp_MByteEn_o      => cpu3_MByteEn_o,
536   --      ocp_SCmdAccept_i   => cpu3_SCmdAccept_i,
537   --      ocp_SResp_i        => cpu3_SResp_i,
538   --      ocp_SData_i        => cpu3_SData_i
539   --    );
540   --
541   --  cpu4 : or1200_ocp
542   --    port map(
543   --      clk_i              => clk,
544   --      rst_i              => reset,
545   --
546   --      -- OCP Master interface signals
547   --      ocp_MCmd_o         => cpu4_MCmd_o,
548   --      ocp_Maddr_o        => cpu4_Maddr_o,
549   --      ocp_MData_o        => cpu4_MData_o,
550   --      ocp_MByteEn_o      => cpu4_MByteEn_o,
551   --      ocp_SCmdAccept_i   => cpu4_SCmdAccept_i,
552   --      ocp_SResp_i        => cpu4_SResp_i,
553   --      ocp_SData_i        => cpu4_SData_i
554   --    );
555
556
557     mem0 : core_mem_ocp
558     port map (
559       clk_i              => clk,
560       rst_i              => reset,
561       ocp_MCmd_i         => mem0_MCmd_i,
562       ocp_MAddr_i        => mem0_MAddr_i,
563       ocp_MData_i        => mem0_MData_i,
564       ocp_MByteEn_i      => mem0_MByteEn_i,
565       ocp_SCmdAccept_o   => mem0_SCmdAccept_o,
566       ocp_SResp_o        => mem0_SResp_o,
567       ocp_SData_o        => mem0_SData_o
568       );
569
570     uart : uart16550_ocp
571     port map(
572       clk_i             => clk,
573       rst_i             => reset,
574       -- OCP slave interface
575       ocp_MCmd_i        => uart_MCmd_i,
576       ocp_Maddr_i       => uart_Maddr_i,
577       ocp_MData_i       => uart_MData_i,
578       ocp_MByteEn_i     => uart_MByteEn_i,
579       ocp_SCmdAccept_o  => uart_SCmdAccept_o,
580       ocp_SResp_o       => uart_SResp_o,
```

```
581       ocp_SData_o       => uart_SData_o,
582       -- uart --
583       int_o             => open,
584       -- RS232 interface
585       tx                => tx,
586       rx                => rx,
587       rts               => open,
588       cts               => '1',
589       dtr               => open,
590       dsr               => '1',
591       ri                => '1',
592       dcd               => '1'
593     );
594
595     semaphore : semaphore_ocp
596     generic map (
597       semaphores => 5
598      )
599     port map(
600       clk_i             => clk,
601       rst_i             => reset,
602       -- OCP Slave interface
603       ocp_MCmd_i        => semaphore_MCmd_i,
604       ocp_Maddr_i       => semaphore_Maddr_i,
605
606       ocp_MData_i       => semaphore_MData_i,
607
608       ocp_MByteEn_i     => semaphore_MByteEn_i,
609       ocp_SCmdAccept_o  => semaphore_SCmdAccept_o,
610       ocp_SResp_o       => semaphore_SResp_o,
611       ocp_SData_o       => semaphore_SData_o
612     );
613
614
615     cpu0_master_na: master_na
616     generic map(
617       routing_table     => cpu0_routing_table
618     )
619     port map(
620       clk_i             => clk2,
621       reset_i           => reset,
622       ocp_MCmd_i        => cpu0_MCmd_o,
623       ocp_Maddr_i       => cpu0_Maddr_o,
624       ocp_MData_i       => cpu0_MData_o,
625       ocp_MByteEn_i     => cpu0_MByteEn_o,
626       ocp_SCmdAccept_o  => cpu0_SCmdAccept_i,
627       ocp_SResp_o       => cpu0_SResp_i,
628       ocp_SData_o       => cpu0_SData_i,
629       rh_out            => cpu0_rh_in,
630       ri_out            => cpu0_ri_in,
631       re_out            => cpu0_re_in,
632       ack_out           => cpu0_ack_in,
633       data_out          => cpu0_data_in,
634       rh_in             => cpu0_rh_out,
635       ri_in             => cpu0_ri_out,
636       re_in             => cpu0_re_out,
637       ack_in            => cpu0_ack_out,
638       data_in           => cpu0_data_out
639     );
640
641     cpu1_master_na: master_na
642     generic map(
643       routing_table     => cpu1_routing_table
644     )
645     port map(
```

```
646       clk_i             => clk2,
647       reset_i           => reset,
648       ocp_MCmd_i        => cpu1_MCmd_o,
649       ocp_Maddr_i       => cpu1_Maddr_o,
650       ocp_MData_i       => cpu1_MData_o,
651       ocp_MByteEn_i     => cpu1_MByteEn_o,
652       ocp_SCmdAccept_o  => cpu1_SCmdAccept_i,
653       ocp_SResp_o       => cpu1_SResp_i,
654       ocp_SData_o       => cpu1_SData_i,
655       rh_out            => cpu1_rh_in,
656       ri_out            => cpu1_ri_in,
657       re_out            => cpu1_re_in,
658       ack_out           => cpu1_ack_in,
659       data_out          => cpu1_data_in,
660       rh_in             => cpu1_rh_out,
661       ri_in             => cpu1_ri_out,
662       re_in             => cpu1_re_out,
663       ack_in            => cpu1_ack_out,
664       data_in           => cpu1_data_out
665     );
666
667     cpu2_master_na: master_na
668     generic map(
669       routing_table     => cpu2_routing_table
670     )
671     port map(
672       clk_i             => clk,
673       reset_i           => reset,
674       ocp_MCmd_i        => cpu2_MCmd_o,
675       ocp_Maddr_i       => cpu2_Maddr_o,
676       ocp_MData_i       => cpu2_MData_o,
677       ocp_MByteEn_i     => cpu2_MByteEn_o,
678       ocp_SCmdAccept_o  => cpu2_SCmdAccept_i,
679       ocp_SResp_o       => cpu2_SResp_i,
680       ocp_SData_o       => cpu2_SData_i,
681       rh_out            => cpu2_rh_in,
682       ri_out            => cpu2_ri_in,
683       re_out            => cpu2_re_in,
684       ack_out           => cpu2_ack_in,
685       data_out          => cpu2_data_in,
686       rh_in             => cpu2_rh_out,
687       ri_in             => cpu2_ri_out,
688       re_in             => cpu2_re_out,
689       ack_in            => cpu2_ack_out,
690       data_in           => cpu2_data_out
691     );
692
693   --  cpu3_master_na: master_na
694   --  generic map(
695   --    routing_table     => cpu3_routing_table
696   --  )
697   --  port map(
698   --    clk_i             => clk,
699   --    reset_i           => reset,
700   --    ocp_MCmd_i        => cpu3_MCmd_o,
701   --    ocp_Maddr_i       => cpu3_Maddr_o,
702   --    ocp_MData_i       => cpu3_MData_o,
703   --    ocp_MByteEn_i     => cpu3_MByteEn_o,
704   --    ocp_SCmdAccept_o  => cpu3_SCmdAccept_i,
705   --    ocp_SResp_o       => cpu3_SResp_i,
706   --    ocp_SData_o       => cpu3_SData_i,
707   --    rh_out            => cpu3_rh_in,
708   --    ri_out            => cpu3_ri_in,
709   --    re_out            => cpu3_re_in,
710   --    ack_out           => cpu3_ack_in,
```

```
711 --    data_out          => cpu3_data_in,
712 --    rh_in             => cpu3_rh_out,
713 --    ri_in             => cpu3_ri_out,
714 --    re_in             => cpu3_re_out,
715 --    ack_in            => cpu3_ack_out,
716 --    data_in           => cpu3_data_out
717 --  );
718 ----
719 --  cpu4_master_na: master_na
720 --  generic map(
721 --    routing_table     => cpu4_routing_table
722 --  )
723 --  port map(
724 --    clk_i             => clk,
725 --    reset_i           => reset,
726 --    ocp_MCmd_i        => cpu4_MCmd_o,
727 --    ocp_Maddr_i       => cpu4_Maddr_o,
728 --    ocp_MData_i       => cpu4_MData_o,
729 --    ocp_MByteEn_i     => cpu4_MByteEn_o,
730 --    ocp_SCmdAccept_o  => cpu4_SCmdAccept_i,
731 --    ocp_SResp_o       => cpu4_SResp_i,
732 --    ocp_SData_o       => cpu4_SData_i,
733 --    rh_out            => cpu4_rh_in,
734 --    ri_out            => cpu4_ri_in,
735 --    re_out            => cpu4_re_in,
736 --    ack_out           => cpu4_ack_in,
737 --    data_out          => cpu4_data_in,
738 --    rh_in             => cpu4_rh_out,
739 --    ri_in             => cpu4_ri_out,
740 --    re_in             => cpu4_re_out,
741 --    ack_in            => cpu4_ack_out,
742 --    data_in           => cpu4_data_out
743 --  );
744
745   mem0_slave_na :slave_na
746   port map(
747     clk_i               => clk,
748     reset_i             => reset,
749     ocp_MCmd_o          => mem0_MCmd_i,
750     ocp_Maddr_o         => mem0_Maddr_i,
751     ocp_MData_o         => mem0_MData_i,
752     ocp_MByteEn_o       => mem0_MByteEn_i,
753     ocp_SCmdAccept_i    => mem0_SCmdAccept_o,
754     ocp_SResp_i         => mem0_SResp_o,
755     ocp_SData_i         => mem0_SData_o,
756     rh_out              => mem0_rh_in,
757     ri_out              => mem0_ri_in,
758     re_out              => mem0_re_in,
759     ack_out             => mem0_ack_in,
760     data_out            => mem0_data_in,
761     rh_in               => mem0_rh_out,
762     ri_in               => mem0_ri_out,
763     re_in               => mem0_re_out,
764     ack_in              => mem0_ack_out,
765     data_in             => mem0_data_out
766   );
767
768   uart_slave_na :slave_na
769   port map(
770     clk_i               => clk,
771     reset_i             => reset,
772     ocp_MCmd_o          => uart_MCmd_i,
773     ocp_Maddr_o         => uart_Maddr_i,
774     ocp_MData_o         => uart_MData_i,
775     ocp_MByteEn_o       => uart_MByteEn_i,
```

```
776       ocp_SCmdAccept_i    => uart_SCmdAccept_o,
777       ocp_SResp_i         => uart_SResp_o,
778       ocp_SData_i         => uart_SData_o,
779       rh_out              => uart_rh_in,
780       ri_out              => uart_ri_in,
781       re_out              => uart_re_in,
782       ack_out             => uart_ack_in,
783       data_out            => uart_data_in,
784       rh_in               => uart_rh_out,
785       ri_in               => uart_ri_out,
786       re_in               => uart_re_out,
787       ack_in              => uart_ack_out,
788       data_in             => uart_data_out
789     );
790
791     semaphore_slave_na :slave_na
792     port map(
793       clk_i               => clk,
794       reset_i             => reset,
795       ocp_MCmd_o          => semaphore_MCmd_i,
796       ocp_Maddr_o         => semaphore_Maddr_i,
797       ocp_MData_o         => semaphore_MData_i,
798       ocp_MByteEn_o       => semaphore_MByteEn_i,
799       ocp_SCmdAccept_i    => semaphore_SCmdAccept_o,
800       ocp_SResp_i         => semaphore_SResp_o,
801       ocp_SData_i         => semaphore_SData_o,
802       rh_out              => semaphore_rh_in,
803       ri_out              => semaphore_ri_in,
804       re_out              => semaphore_re_in,
805       ack_out             => semaphore_ack_in,
806       data_out            => semaphore_data_in,
807       rh_in               => semaphore_rh_out,
808       ri_in               => semaphore_ri_out,
809       re_in               => semaphore_re_out,
810       ack_in              => semaphore_ack_out,
811       data_in             => semaphore_data_out
812     );
813
814     mesh : noc_mesh
815       port map(
816         reset       => reset,
817
818         r0_rh_in    => cpu0_rh_in,
819         r0_ri_in    => cpu0_ri_in,
820         r0_re_in    => cpu0_re_in,
821         r0_ack_in   => cpu0_ack_in,
822         r0_data_in  => cpu0_data_in,
823         r0_rh_out   => cpu0_rh_out,
824         r0_ri_out   => cpu0_ri_out,
825         r0_re_out   => cpu0_re_out,
826         r0_ack_out  => cpu0_ack_out,
827         r0_data_out => cpu0_data_out,
828
829         r1_rh_in    => cpu1_rh_in,
830         r1_ri_in    => cpu1_ri_in,
831         r1_re_in    => cpu1_re_in,
832         r1_ack_in   => cpu1_ack_in,
833         r1_data_in  => cpu1_data_in,
834         r1_rh_out   => cpu1_rh_out,
835         r1_ri_out   => cpu1_ri_out,
836         r1_re_out   => cpu1_re_out,
837         r1_ack_out  => cpu1_ack_out,
838         r1_data_out => cpu1_data_out,
839
840   --      r1_rh_in    => '0',
```

```
841 --       r1_ri_in     => '0',
842 --       r1_re_in     => '0',
843 --       r1_ack_in    => open,
844 --       r1_data_in   => FLIT_ZERO,
845 --       r1_rh_out    => open,
846 --       r1_ri_out    => open,
847 --       r1_re_out    => open,
848 --       r1_ack_out   => '0',
849 --       r1_data_out  => open,
850
851         r2_rh_in     => mem0_rh_in,
852         r2_ri_in     => mem0_ri_in,
853         r2_re_in     => mem0_re_in,
854         r2_ack_in    => mem0_ack_in,
855         r2_data_in   => mem0_data_in,
856         r2_rh_out    => mem0_rh_out,
857         r2_ri_out    => mem0_ri_out,
858         r2_re_out    => mem0_re_out,
859         r2_ack_out   => mem0_ack_out,
860         r2_data_out  => mem0_data_out,
861
862         r3_rh_in     => cpu2_rh_in,
863         r3_ri_in     => cpu2_ri_in,
864         r3_re_in     => cpu2_re_in,
865         r3_ack_in    => cpu2_ack_in,
866         r3_data_in   => cpu2_data_in,
867         r3_rh_out    => cpu2_rh_out,
868         r3_ri_out    => cpu2_ri_out,
869         r3_re_out    => cpu2_re_out,
870         r3_ack_out   => cpu2_ack_out,
871         r3_data_out  => cpu2_data_out,
872
873 --       r3_rh_in     => '0',
874 --       r3_ri_in     => '0',
875 --       r3_re_in     => '0',
876 --       r3_ack_in    => open,
877 --       r3_data_in   => FLIT_ZERO,
878 --       r3_rh_out    => open,
879 --       r3_ri_out    => open,
880 --       r3_re_out    => open,
881 --       r3_ack_out   => '0',
882 --       r3_data_out  => open,
883
884 --       r4_rh_in     => cpu3_rh_in,
885 --       r4_ri_in     => cpu3_ri_in,
886 --       r4_re_in     => cpu3_re_in,
887 --       r4_ack_in    => cpu3_ack_in,
888 --       r4_data_in   => cpu3_data_in,
889 --       r4_rh_out    => cpu3_rh_out,
890 --       r4_ri_out    => cpu3_ri_out,
891 --       r4_re_out    => cpu3_re_out,
892 --       r4_ack_out   => cpu3_ack_out,
893 --       r4_data_out  => cpu3_data_out,
894
895         r4_rh_in     => '0',
896         r4_ri_in     => '0',
897         r4_re_in     => '0',
898         r4_ack_in    => open,
899         r4_data_in   => FLIT_ZERO,
900         r4_rh_out    => open,
901         r4_ri_out    => open,
902         r4_re_out    => open,
903         r4_ack_out   => '0',
904         r4_data_out  => open,
905
```

```
906         r5_rh_in    => uart_rh_in,
907         r5_ri_in    => uart_ri_in,
908         r5_re_in    => uart_re_in,
909         r5_ack_in   => uart_ack_in,
910         r5_data_in  => uart_data_in,
911         r5_rh_out   => uart_rh_out,
912         r5_ri_out   => uart_ri_out,
913         r5_re_out   => uart_re_out,
914         r5_ack_out  => uart_ack_out,
915         r5_data_out => uart_data_out,
916
917         r5_east_rh_in     => semaphore_rh_in,
918         r5_east_ri_in     => semaphore_ri_in,
919         r5_east_re_in     => semaphore_re_in,
920         r5_east_ack_in    => semaphore_ack_in,
921         r5_east_data_in   => semaphore_data_in,
922         r5_east_rh_out    => semaphore_rh_out,
923         r5_east_ri_out    => semaphore_ri_out,
924         r5_east_re_out    => semaphore_re_out,
925         r5_east_ack_out   => semaphore_ack_out,
926         r5_east_data_out  => semaphore_data_out,
927
928         r6_rh_in    => '0',
929         r6_ri_in    => '0',
930         r6_re_in    => '0',
931         r6_ack_in   => open,
932         r6_data_in  => FLIT_ZERO,
933         r6_rh_out   => open,
934         r6_ri_out   => open,
935         r6_re_out   => open,
936         r6_ack_out  => '0',
937         r6_data_out => open,
938
939 --      r7_rh_in    => cpu4_rh_in,
940 --      r7_ri_in    => cpu4_ri_in,
941 --      r7_re_in    => cpu4_re_in,
942 --      r7_ack_in   => cpu4_ack_in,
943 --      r7_data_in  => cpu4_data_in,
944 --      r7_rh_out   => cpu4_rh_out,
945 --      r7_ri_out   => cpu4_ri_out,
946 --      r7_re_out   => cpu4_re_out,
947 --      r7_ack_out  => cpu4_ack_out,
948 --      r7_data_out => cpu4_data_out,
949
950         r7_rh_in    => '0',
951         r7_ri_in    => '0',
952         r7_re_in    => '0',
953         r7_ack_in   => open,
954         r7_data_in  => FLIT_ZERO,
955         r7_rh_out   => open,
956         r7_ri_out   => open,
957         r7_re_out   => open,
958         r7_ack_out  => '0',
959         r7_data_out => open,
960
961         r8_rh_in    => '0',
962         r8_ri_in    => '0',
963         r8_re_in    => '0',
964         r8_ack_in   => open,
965         r8_data_in  => FLIT_ZERO,
966         r8_rh_out   => open,
967         r8_ri_out   => open,
968         r8_re_out   => open,
969         r8_ack_out  => '0',
970         r8_data_out => open
```

```
971     );
972
973 end struct;
```

A.5.5.2 noc_mesh.vhd

```
  1 library IEEE;
  2 use IEEE.STD_LOGIC_1164.ALL;
  3 use IEEE.STD_LOGIC_ARITH.ALL;
  4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
  5 use work.types.all;
  6
  7 entity noc_mesh is
  8
  9     port(
 10       reset : in std_logic;
 11
 12       -- router0 ports
 13       --input
 14       r0_rh_in    : in  std_logic;
 15       r0_ri_in    : in  std_logic;
 16       r0_re_in    : in  std_logic;
 17       r0_ack_in   : out std_logic;
 18       r0_data_in  : in  flit_data;
 19       --output
 20       r0_rh_out   : out std_logic;
 21       r0_ri_out   : out std_logic;
 22       r0_re_out   : out std_logic;
 23       r0_ack_out  : in  std_logic;
 24       r0_data_out : out flit_data;
 25
 26       -- router1 ports
 27       --input
 28       r1_rh_in    : in  std_logic;
 29       r1_ri_in    : in  std_logic;
 30       r1_re_in    : in  std_logic;
 31       r1_ack_in   : out std_logic;
 32       r1_data_in  : in  flit_data;
 33       --output
 34       r1_rh_out   : out std_logic;
 35       r1_ri_out   : out std_logic;
 36       r1_re_out   : out std_logic;
 37       r1_ack_out  : in  std_logic;
 38       r1_data_out : out flit_data;
 39
 40       -- router2 ports
 41       --input
 42       r2_rh_in    : in  std_logic;
 43       r2_ri_in    : in  std_logic;
 44       r2_re_in    : in  std_logic;
 45       r2_ack_in   : out std_logic;
 46       r2_data_in  : in  flit_data;
 47       --output
 48       r2_rh_out   : out std_logic;
 49       r2_ri_out   : out std_logic;
 50       r2_re_out   : out std_logic;
 51       r2_ack_out  : in  std_logic;
 52       r2_data_out : out flit_data;
 53
 54       -- router3 ports
 55       --input
 56       r3_rh_in    : in  std_logic;
 57       r3_ri_in    : in  std_logic;
```

```
58        r3_re_in    : in  std_logic;
59        r3_ack_in   : out std_logic;
60        r3_data_in  : in  flit_data;
61        --output
62        r3_rh_out   : out std_logic;
63        r3_ri_out   : out std_logic;
64        r3_re_out   : out std_logic;
65        r3_ack_out  : in  std_logic;
66        r3_data_out : out flit_data;
67
68        -- router4 ports
69        --input
70        r4_rh_in    : in  std_logic;
71        r4_ri_in    : in  std_logic;
72        r4_re_in    : in  std_logic;
73        r4_ack_in   : out std_logic;
74        r4_data_in  : in  flit_data;
75        --output
76        r4_rh_out   : out std_logic;
77        r4_ri_out   : out std_logic;
78        r4_re_out   : out std_logic;
79        r4_ack_out  : in  std_logic;
80        r4_data_out : out flit_data;
81
82        -- router5 ports
83        --input
84        r5_rh_in    : in  std_logic;
85        r5_ri_in    : in  std_logic;
86        r5_re_in    : in  std_logic;
87        r5_ack_in   : out std_logic;
88        r5_data_in  : in  flit_data;
89        --output
90        r5_rh_out   : out std_logic;
91        r5_ri_out   : out std_logic;
92        r5_re_out   : out std_logic;
93        r5_ack_out  : in  std_logic;
94        r5_data_out : out flit_data;
95
96        --input
97        r5_east_rh_in     : in  std_logic;
98        r5_east_ri_in     : in  std_logic;
99        r5_east_re_in     : in  std_logic;
100       r5_east_ack_in    : out std_logic;
101       r5_east_data_in   : in  flit_data;
102       --output
103       r5_east_rh_out    : out std_logic;
104       r5_east_ri_out    : out std_logic;
105       r5_east_re_out    : out std_logic;
106       r5_east_ack_out   : in  std_logic;
107       r5_east_data_out  : out flit_data;
108
109       -- router6 ports
110       --input
111       r6_rh_in    : in  std_logic;
112       r6_ri_in    : in  std_logic;
113       r6_re_in    : in  std_logic;
114       r6_ack_in   : out std_logic;
115       r6_data_in  : in  flit_data;
116       --output
117       r6_rh_out   : out std_logic;
118       r6_ri_out   : out std_logic;
119       r6_re_out   : out std_logic;
120       r6_ack_out  : in  std_logic;
121       r6_data_out : out flit_data;
122
```

```
123         -- router7 ports
124         --input
125         r7_rh_in    : in  std_logic;
126         r7_ri_in    : in  std_logic;
127         r7_re_in    : in  std_logic;
128         r7_ack_in   : out std_logic;
129         r7_data_in  : in  flit_data;
130         --output
131         r7_rh_out   : out std_logic;
132         r7_ri_out   : out std_logic;
133         r7_re_out   : out std_logic;
134         r7_ack_out  : in  std_logic;
135         r7_data_out : out flit_data;
136
137         -- router8 ports
138         --input
139         r8_rh_in    : in  std_logic;
140         r8_ri_in    : in  std_logic;
141         r8_re_in    : in  std_logic;
142         r8_ack_in   : out std_logic;
143         r8_data_in  : in  flit_data;
144         --output
145         r8_rh_out   : out std_logic;
146         r8_ri_out   : out std_logic;
147         r8_re_out   : out std_logic;
148         r8_ack_out  : in  std_logic;
149         r8_data_out : out flit_data
150       );
151
152   end noc_mesh;
153
154   architecture struct of noc_mesh is
155
156     component be_router is
157       port(
158         reset : in std_logic;
159         -- Input ports --
160         north_rh_in   : in  std_logic;
161         north_ri_in   : in  std_logic;
162         north_re_in   : in  std_logic;
163         north_ack_in  : out std_logic;
164         north_data_in : in  flit_data;
165
166         west_rh_in    : in  std_logic;
167         west_ri_in    : in  std_logic;
168         west_re_in    : in  std_logic;
169         west_ack_in   : out std_logic;
170         west_data_in  : in  flit_data;
171
172         south_rh_in   : in  std_logic;
173         south_ri_in   : in  std_logic;
174         south_re_in   : in  std_logic;
175         south_ack_in  : out std_logic;
176         south_data_in : in  flit_data;
177
178         east_rh_in    : in  std_logic;
179         east_ri_in    : in  std_logic;
180         east_re_in    : in  std_logic;
181         east_ack_in   : out std_logic;
182         east_data_in  : in  flit_data;
183
184         local_rh_in   : in  std_logic;
185         local_ri_in   : in  std_logic;
186         local_re_in   : in  std_logic;
187         local_ack_in  : out std_logic;
```

```
188         local_data_in : in  flit_data;
189
190         -- Output ports --
191         north_rh_out    : out std_logic;
192         north_ri_out    : out std_logic;
193         north_re_out    : out std_logic;
194         north_ack_out   : in  std_logic;
195         north_data_out  : out flit_data;
196
197         west_rh_out     : out std_logic;
198         west_ri_out     : out std_logic;
199         west_re_out     : out std_logic;
200         west_ack_out    : in  std_logic;
201         west_data_out   : out flit_data;
202
203         south_rh_out    : out std_logic;
204         south_ri_out    : out std_logic;
205         south_re_out    : out std_logic;
206         south_ack_out   : in  std_logic;
207         south_data_out  : out flit_data;
208
209         east_rh_out     : out std_logic;
210         east_ri_out     : out std_logic;
211         east_re_out     : out std_logic;
212         east_ack_out    : in  std_logic;
213         east_data_out   : out flit_data;
214
215         local_rh_out    : out std_logic;
216         local_ri_out    : out std_logic;
217         local_re_out    : out std_logic;
218         local_ack_out   : in  std_logic;
219         local_data_out  : out flit_data
220       );
221     end component;
222
223     -- hs signals
224
225     signal  r3_north_rh,  r3_north_ri,  r3_north_re,  r3_north_ack,
226             r4_north_rh,  r4_north_ri,  r4_north_re,  r4_north_ack,
227             r5_north_rh,  r5_north_ri,  r5_north_re,  r5_north_ack,
228             r6_north_rh,  r6_north_ri,  r6_north_re,  r6_north_ack,
229             r7_north_rh,  r7_north_ri,  r7_north_re,  r7_north_ack,
230             r8_north_rh,  r8_north_ri,  r8_north_re,  r8_north_ack,
231
232             r0_east_rh,   r0_east_ri,   r0_east_re,   r0_east_ack,
233             r1_east_rh,   r1_east_ri,   r1_east_re,   r1_east_ack,
234             r3_east_rh,   r3_east_ri,   r3_east_re,   r3_east_ack,
235             r4_east_rh,   r4_east_ri,   r4_east_re,   r4_east_ack,
236             r6_east_rh,   r6_east_ri,   r6_east_re,   r6_east_ack,
237             r7_east_rh,   r7_east_ri,   r7_east_re,   r7_east_ack,
238
239             r0_south_rh,  r0_south_ri,  r0_south_re,  r0_south_ack,
240             r1_south_rh,  r1_south_ri,  r1_south_re,  r1_south_ack,
241             r2_south_rh,  r2_south_ri,  r2_south_re,  r2_south_ack,
242             r3_south_rh,  r3_south_ri,  r3_south_re,  r3_south_ack,
243             r4_south_rh,  r4_south_ri,  r4_south_re,  r4_south_ack,
244             r5_south_rh,  r5_south_ri,  r5_south_re,  r5_south_ack,
245
246             r1_west_rh,   r1_west_ri,   r1_west_re,   r1_west_ack,
247             r2_west_rh,   r2_west_ri,   r2_west_re,   r2_west_ack,
248             r4_west_rh,   r4_west_ri,   r4_west_re,   r4_west_ack,
249             r5_west_rh,   r5_west_ri,   r5_west_re,   r5_west_ack,
250             r7_west_rh,   r7_west_ri,   r7_west_re,   r7_west_ack,
251             r8_west_rh,   r8_west_ri,   r8_west_re,   r8_west_ack   : std_logic
                    ;
```

```
252
253   -- data signals
254   signal  r3_north_data,  r4_north_data,  r5_north_data,  r6_north_data,
255           r7_north_data,  r8_north_data,  r0_east_data,   r1_east_data,
256           r3_east_data,   r4_east_data,   r6_east_data,   r7_east_data,
257           r0_south_data,  r1_south_data,  r2_south_data,  r3_south_data,
258           r4_south_data,  r5_south_data,  r1_west_data,   r2_west_data,
259           r4_west_data,   r5_west_data,   r7_west_data,   r8_west_data  :
                flit_data;
260
261
262 begin
263   --  Mesh layout
264   --
265   --  r0 ---- r1 ---- r2
266   --  |       |       |
267   --  |       |       |
268   --  r3 ---- r4 ---- r5
269   --  |       |       |
270   --  |       |       |
271   --  r6 ---- r7 ---- r8
272
273
274   r0 : be_router
275     port map(
276       reset         => reset,
277
278       north_rh_in   => '0',
279       north_ri_in   => '0',
280       north_re_in   => '0',
281       north_ack_in  => open,
282       north_data_in => (others => '0'),
283
284       west_rh_in    => '0',
285       west_ri_in    => '0',
286       west_re_in    => '0',
287       west_ack_in   => open,
288       west_data_in  => (others => '0'),
289
290       south_rh_in   => r3_north_rh,
291       south_ri_in   => r3_north_ri,
292       south_re_in   => r3_north_re,
293       south_ack_in  => r3_north_ack,
294       south_data_in => r3_north_data,
295
296       east_rh_in    => r1_west_rh,
297       east_ri_in    => r1_west_ri,
298       east_re_in    => r1_west_re,
299       east_ack_in   => r1_west_ack,
300       east_data_in  => r1_west_data,
301
302       local_rh_in   => r0_rh_in,
303       local_ri_in   => r0_ri_in,
304       local_re_in   => r0_re_in,
305       local_ack_in  => r0_ack_in,
306       local_data_in => r0_data_in,
307
308       -- Output ports --
309       north_rh_out    => open,
310       north_ri_out    => open,
311       north_re_out    => open,
312       north_ack_out   => '0',
313       north_data_out  => open,
314
315       west_rh_out     => open,
```

```
316         west_ri_out     => open,
317         west_re_out     => open,
318         west_ack_out    => '0',
319         west_data_out   => open,
320
321         south_rh_out    => r0_south_rh,
322         south_ri_out    => r0_south_ri,
323         south_re_out    => r0_south_re,
324         south_ack_out   => r0_south_ack,
325         south_data_out  => r0_south_data,
326
327         east_rh_out     => r0_east_rh,
328         east_ri_out     => r0_east_ri,
329         east_re_out     => r0_east_re,
330         east_ack_out    => r0_east_ack,
331         east_data_out   => r0_east_data,
332
333         local_rh_out    => r0_rh_out,
334         local_ri_out    => r0_ri_out,
335         local_re_out    => r0_re_out,
336         local_ack_out   => r0_ack_out,
337         local_data_out  => r0_data_out
338       );
339
340       r1 : be_router
341       port map(
342         reset         => reset,
343
344         north_rh_in   => '0',
345         north_ri_in   => '0',
346         north_re_in   => '0',
347         north_ack_in  => open,
348         north_data_in => (others => '0'),
349
350         west_rh_in    => r0_east_rh,
351         west_ri_in    => r0_east_ri,
352         west_re_in    => r0_east_re,
353         west_ack_in   => r0_east_ack,
354         west_data_in  => r0_east_data,
355
356         south_rh_in   => r4_north_rh,
357         south_ri_in   => r4_north_ri,
358         south_re_in   => r4_north_re,
359         south_ack_in  => r4_north_ack,
360         south_data_in => r4_north_data,
361
362         east_rh_in    => r2_west_rh,
363         east_ri_in    => r2_west_ri,
364         east_re_in    => r2_west_re,
365         east_ack_in   => r2_west_ack,
366         east_data_in  => r2_west_data,
367
368         local_rh_in   => r1_rh_in,
369         local_ri_in   => r1_ri_in,
370         local_re_in   => r1_re_in,
371         local_ack_in  => r1_ack_in,
372         local_data_in => r1_data_in,
373
374         -- Output ports --
375         north_rh_out    => open,
376         north_ri_out    => open,
377         north_re_out    => open,
378         north_ack_out   => '0',
379         north_data_out  => open,
380
```

```
381         west_rh_out     => r1_west_rh,
382         west_ri_out     => r1_west_ri,
383         west_re_out     => r1_west_re,
384         west_ack_out    => r1_west_ack,
385         west_data_out   => r1_west_data,
386
387         south_rh_out    => r1_south_rh,
388         south_ri_out    => r1_south_ri,
389         south_re_out    => r1_south_re,
390         south_ack_out   => r1_south_ack,
391         south_data_out  => r1_south_data,
392
393         east_rh_out     => r1_east_rh,
394         east_ri_out     => r1_east_ri,
395         east_re_out     => r1_east_re,
396         east_ack_out    => r1_east_ack,
397         east_data_out   => r1_east_data,
398
399         local_rh_out    => r1_rh_out,
400         local_ri_out    => r1_ri_out,
401         local_re_out    => r1_re_out,
402         local_ack_out   => r1_ack_out,
403         local_data_out  => r1_data_out
404       );
405
406       r2 : be_router
407       port map(
408         reset         => reset,
409
410         north_rh_in   => '0',
411         north_ri_in   => '0',
412         north_re_in   => '0',
413         north_ack_in  => open,
414         north_data_in => (others => '0'),
415
416         west_rh_in    => r1_east_rh,
417         west_ri_in    => r1_east_ri,
418         west_re_in    => r1_east_re,
419         west_ack_in   => r1_east_ack,
420         west_data_in  => r1_east_data,
421
422         south_rh_in   => r5_north_rh,
423         south_ri_in   => r5_north_ri,
424         south_re_in   => r5_north_re,
425         south_ack_in  => r5_north_ack,
426         south_data_in => r5_north_data,
427
428         east_rh_in    => '0',
429         east_ri_in    => '0',
430         east_re_in    => '0',
431         east_ack_in   => open,
432         east_data_in  => (others => '0'),
433
434         local_rh_in   => r2_rh_in,
435         local_ri_in   => r2_ri_in,
436         local_re_in   => r2_re_in,
437         local_ack_in  => r2_ack_in,
438         local_data_in => r2_data_in,
439
440         -- Output ports --
441         north_rh_out    => open,
442         north_ri_out    => open,
443         north_re_out    => open,
444         north_ack_out   => '0',
445         north_data_out  => open,
```

```
446
447         west_rh_out     => r2_west_rh,
448         west_ri_out     => r2_west_ri,
449         west_re_out     => r2_west_re,
450         west_ack_out    => r2_west_ack,
451         west_data_out   => r2_west_data,
452
453         south_rh_out    => r2_south_rh,
454         south_ri_out    => r2_south_ri,
455         south_re_out    => r2_south_re,
456         south_ack_out   => r2_south_ack,
457         south_data_out  => r2_south_data,
458
459         east_rh_out     => open,
460         east_ri_out     => open,
461         east_re_out     => open,
462         east_ack_out    => '0',
463         east_data_out   => open,
464
465         local_rh_out    => r2_rh_out,
466         local_ri_out    => r2_ri_out,
467         local_re_out    => r2_re_out,
468         local_ack_out   => r2_ack_out,
469         local_data_out  => r2_data_out
470       );
471
472       r3 : be_router
473       port map(
474         reset         => reset,
475
476         north_rh_in   => r0_south_rh,
477         north_ri_in   => r0_south_ri,
478         north_re_in   => r0_south_re,
479         north_ack_in  => r0_south_ack,
480         north_data_in => r0_south_data,
481
482         west_rh_in    => '0',
483         west_ri_in    => '0',
484         west_re_in    => '0',
485         west_ack_in   => open,
486         west_data_in  => (others => '0'),
487
488 --        south_rh_in   => r6_north_rh,
489 --        south_ri_in   => r6_north_ri,
490 --        south_re_in   => r6_north_re,
491 --        south_ack_in  => r6_north_ack,
492 --        south_data_in => r6_north_data,
493
494         south_rh_in   => '0',
495         south_ri_in   => '0',
496         south_re_in   => '0',
497         south_ack_in  => open,
498         south_data_in => (others => '0'),
499
500
501         east_rh_in    => r4_west_rh,
502         east_ri_in    => r4_west_ri,
503         east_re_in    => r4_west_re,
504         east_ack_in   => r4_west_ack,
505         east_data_in  => r4_west_data,
506
507         local_rh_in   => r3_rh_in,
508         local_ri_in   => r3_ri_in,
509         local_re_in   => r3_re_in,
510         local_ack_in  => r3_ack_in,
```

```
511         local_data_in => r3_data_in,
512
513         -- Output ports --
514         north_rh_out    => r3_north_rh,
515         north_ri_out    => r3_north_ri,
516         north_re_out    => r3_north_re,
517         north_ack_out   => r3_north_ack,
518         north_data_out  => r3_north_data,
519
520         west_rh_out     => open,
521         west_ri_out     => open,
522         west_re_out     => open,
523         west_ack_out    => '0',
524         west_data_out   => open,
525
526         south_rh_out    => r3_south_rh,
527         south_ri_out    => r3_south_ri,
528         south_re_out    => r3_south_re,
529         south_ack_out   => r3_south_ack,
530         south_data_out  => r3_south_data,
531
532         east_rh_out     => r3_east_rh,
533         east_ri_out     => r3_east_ri,
534         east_re_out     => r3_east_re,
535         east_ack_out    => r3_east_ack,
536         east_data_out   => r3_east_data,
537
538         local_rh_out    => r3_rh_out,
539         local_ri_out    => r3_ri_out,
540         local_re_out    => r3_re_out,
541         local_ack_out   => r3_ack_out,
542         local_data_out  => r3_data_out
543       );
544
545   --  r3_north_rh <= '0';
546   --  r3_north_ri <= '0';
547   --  r3_north_re <= '0';
548   --  r3_north_data <= (others => '0');
549   --
550   --  r3_south_rh <= '0';
551   --  r3_south_ri <= '0';
552   --  r3_south_re <= '0';
553   --  r3_south_data <= (others => '0');
554   --
555   --  r3_east_rh <= '0';
556   --  r3_east_ri <= '0';
557   --  r3_east_re <= '0';
558   --  r3_east_data<= (others => '0');
559   --
560   --  r3_rh_out <= '0';
561   --  r3_ri_out <= '0';
562   --  r3_re_out <= '0';
563   --  r3_data_out <= (others => '0');
564
565
566       r4 : be_router
567       generic map(
568         enable_north_port => true,
569         enable_east_port  => true,
570         enable_south_port => true,
571         enable_west_port  => true,
572         enable_local_port => true
573       )
574       port map(
575         reset         => reset,
```

```
576
577         north_rh_in   => r1_south_rh,
578         north_ri_in   => r1_south_ri,
579         north_re_in   => r1_south_re,
580         north_ack_in  => r1_south_ack,
581         north_data_in => r1_south_data,
582
583         west_rh_in    => r3_east_rh,
584         west_ri_in    => r3_east_ri,
585         west_re_in    => r3_east_re,
586         west_ack_in   => r3_east_ack,
587         west_data_in  => r3_east_data,
588
589 --        south_rh_in   => r7_north_rh,
590 --        south_ri_in   => r7_north_ri,
591 --        south_re_in   => r7_north_re,
592 --        south_ack_in  => r7_north_ack,
593 --        south_data_in => r7_north_data,
594
595         south_rh_in   => '0',
596         south_ri_in   => '0',
597         south_re_in   => '0',
598         south_ack_in  => open,
599         south_data_in => (others => '0'),
600
601         east_rh_in    => r5_west_rh,
602         east_ri_in    => r5_west_ri,
603         east_re_in    => r5_west_re,
604         east_ack_in   => r5_west_ack,
605         east_data_in  => r5_west_data,
606
607         local_rh_in   => r4_rh_in,
608         local_ri_in   => r4_ri_in,
609         local_re_in   => r4_re_in,
610         local_ack_in  => r4_ack_in,
611         local_data_in => r4_data_in,
612
613         -- Output ports --
614         north_rh_out    => r4_north_rh,
615         north_ri_out    => r4_north_ri,
616         north_re_out    => r4_north_re,
617         north_ack_out   => r4_north_ack,
618         north_data_out  => r4_north_data,
619
620         west_rh_out     => r4_west_rh,
621         west_ri_out     => r4_west_ri,
622         west_re_out     => r4_west_re,
623         west_ack_out    => r4_west_ack,
624         west_data_out   => r4_west_data,
625
626         south_rh_out    => r4_south_rh,
627         south_ri_out    => r4_south_ri,
628         south_re_out    => r4_south_re,
629         south_ack_out   => r4_south_ack,
630         south_data_out  => r4_south_data,
631
632         east_rh_out     => r4_east_rh,
633         east_ri_out     => r4_east_ri,
634         east_re_out     => r4_east_re,
635         east_ack_out    => r4_east_ack,
636         east_data_out   => r4_east_data,
637
638         local_rh_out    => r4_rh_out,
639         local_ri_out    => r4_ri_out,
640         local_re_out    => r4_re_out,
```

```
641         local_ack_out   => r4_ack_out,
642         local_data_out  => r4_data_out
643       );
644
645       r5 : be_router
646       generic map(
647         enable_north_port => true,
648         enable_east_port  => false,
649         enable_south_port => true,
650         enable_west_port  => true,
651         enable_local_port => true
652       )
653       port map(
654         reset         => reset,
655
656         north_rh_in   => r2_south_rh,
657         north_ri_in   => r2_south_ri,
658         north_re_in   => r2_south_re,
659         north_ack_in  => r2_south_ack,
660         north_data_in => r2_south_data,
661
662         west_rh_in    => r4_east_rh,
663         west_ri_in    => r4_east_ri,
664         west_re_in    => r4_east_re,
665         west_ack_in   => r4_east_ack,
666         west_data_in  => r4_east_data,
667
668 --        south_rh_in   => r8_north_rh,
669 --        south_ri_in   => r8_north_ri,
670 --        south_re_in   => r8_north_re,
671 --        south_ack_in  => r8_north_ack,
672 --        south_data_in => r8_north_data,
673
674         south_rh_in   => '0',
675         south_ri_in   => '0',
676         south_re_in   => '0',
677         south_ack_in  => open,
678         south_data_in => (others => '0'),
679
680         east_rh_in    => r5_east_rh_in,
681         east_ri_in    => r5_east_ri_in,
682         east_re_in    => r5_east_re_in,
683         east_ack_in   => r5_east_ack_in,
684         east_data_in  => r5_east_data_in,
685
686         local_rh_in   => r5_rh_in,
687         local_ri_in   => r5_ri_in,
688         local_re_in   => r5_re_in,
689         local_ack_in  => r5_ack_in,
690         local_data_in => r5_data_in,
691
692         -- Output ports --
693         north_rh_out    => r5_north_rh,
694         north_ri_out    => r5_north_ri,
695         north_re_out    => r5_north_re,
696         north_ack_out   => r5_north_ack,
697         north_data_out  => r5_north_data,
698
699         west_rh_out     => r5_west_rh,
700         west_ri_out     => r5_west_ri,
701         west_re_out     => r5_west_re,
702         west_ack_out    => r5_west_ack,
703         west_data_out   => r5_west_data,
704
705         south_rh_out    => r5_south_rh,
```

```
706         south_ri_out    => r5_south_ri,
707         south_re_out    => r5_south_re,
708         south_ack_out   => r5_south_ack,
709         south_data_out  => r5_south_data,
710
711         east_rh_out     => r5_east_rh_out,
712         east_ri_out     => r5_east_ri_out,
713         east_re_out     => r5_east_re_out,
714         east_ack_out    => r5_east_ack_out,
715         east_data_out   => r5_east_data_out,
716
717         local_rh_out    => r5_rh_out,
718         local_ri_out    => r5_ri_out,
719         local_re_out    => r5_re_out,
720         local_ack_out   => r5_ack_out,
721         local_data_out  => r5_data_out
722       );
723
724   --  r6 : be_router
725   --    port map(
726   --       reset          => reset,
727   --
728   --       north_rh_in    => r3_south_rh,
729   --       north_ri_in    => r3_south_ri,
730   --       north_re_in    => r3_south_re,
731   --       north_ack_in   => r3_south_ack,
732   --       north_data_in  => r3_south_data,
733   --
734   --       west_rh_in     => '0',
735   --       west_ri_in     => '0',
736   --       west_re_in     => '0',
737   --       west_ack_in    => open,
738   --       west_data_in   => FLIT_ZERO,
739   --
740   --       south_rh_in    => '0',
741   --       south_ri_in    => '0',
742   --       south_re_in    => '0',
743   --       south_ack_in   => open,
744   --       south_data_in  => FLIT_ZERO,
745   --
746   --       east_rh_in     => r7_west_rh,
747   --       east_ri_in     => r7_west_ri,
748   --       east_re_in     => r7_west_re,
749   --       east_ack_in    => r7_west_ack,
750   --       east_data_in   => r7_west_data,
751   --
752   --       local_rh_in    => r6_rh_in,
753   --       local_ri_in    => r6_ri_in,
754   --       local_re_in    => r6_re_in,
755   --       local_ack_in   => r6_ack_in,
756   --       local_data_in  => r6_data_in,
757   --
758   --       -- Output ports --
759   --       north_rh_out     => r6_north_rh,
760   --       north_ri_out     => r6_north_ri,
761   --       north_re_out     => r6_north_re,
762   --       north_ack_out    => r6_north_ack,
763   --       north_data_out   => r6_north_data,
764   --
765   --       west_rh_out      => open,
766   --       west_ri_out      => open,
767   --       west_re_out      => open,
768   --       west_ack_out     => '0',
769   --       west_data_out    => open,
770   --
```

```
771  --       south_rh_out     => open,
772  --       south_ri_out     => open,
773  --       south_re_out     => open,
774  --       south_ack_out    => '0',
775  --       south_data_out   => open,
776  --
777  --       east_rh_out      => r6_east_rh,
778  --       east_ri_out      => r6_east_ri,
779  --       east_re_out      => r6_east_re,
780  --       east_ack_out     => r6_east_ack,
781  --       east_data_out    => r6_east_data,
782  --
783  --       local_rh_out     => r6_rh_out,
784  --       local_ri_out     => r6_ri_out,
785  --       local_re_out     => r6_re_out,
786  --       local_ack_out    => r6_ack_out,
787  --       local_data_out   => r6_data_out
788  --     );
789  --
790  ----   r6_north_rh <= '0';
791  ----   r6_north_ri <= '0';
792  ----   r6_north_re <= '0';
793  ----   r6_north_data <= (others => '0');
794  ----
795  ----   r6_east_rh <= '0';
796  ----   r6_east_ri <= '0';
797  ----   r6_east_re <= '0';
798  ----   r6_east_data<= (others => '0');
799  ----
800  ----   r6_rh_out <= '0';
801  ----   r6_ri_out <= '0';
802  ----   r6_re_out <= '0';
803  ----   r6_data_out <= (others => '0');
804  --
805  --     r7 : be_router
806  --     port map(
807  --       reset          => reset,
808  --
809  --       north_rh_in    => r4_south_rh,
810  --       north_ri_in    => r4_south_ri,
811  --       north_re_in    => r4_south_re,
812  --       north_ack_in   => r4_south_ack,
813  --       north_data_in  => r4_south_data,
814  --
815  --       west_rh_in     => r6_east_rh,
816  --       west_ri_in     => r6_east_ri,
817  --       west_re_in     => r6_east_re,
818  --       west_ack_in    => r6_east_ack,
819  --       west_data_in   => r6_east_data,
820  --
821  --       south_rh_in    => '0',
822  --       south_ri_in    => '0',
823  --       south_re_in    => '0',
824  --       south_ack_in   => open,
825  --       south_data_in  => FLIT_ZERO,
826  --
827  --       east_rh_in     => r8_west_rh,
828  --       east_ri_in     => r8_west_ri,
829  --       east_re_in     => r8_west_re,
830  --       east_ack_in    => r8_west_ack,
831  --       east_data_in   => r8_west_data,
832  --
833  --       local_rh_in    => r7_rh_in,
834  --       local_ri_in    => r7_ri_in,
835  --       local_re_in    => r7_re_in,
```

```
836 --       local_ack_in  => r7_ack_in,
837 --       local_data_in => r7_data_in,
838 --
839 --       -- Output ports --
840 --       north_rh_out    => r7_north_rh,
841 --       north_ri_out    => r7_north_ri,
842 --       north_re_out    => r7_north_re,
843 --       north_ack_out   => r7_north_ack,
844 --       north_data_out  => r7_north_data,
845 --
846 --       west_rh_out     => r7_west_rh,
847 --       west_ri_out     => r7_west_ri,
848 --       west_re_out     => r7_west_re,
849 --       west_ack_out    => r7_west_ack,
850 --       west_data_out   => r7_west_data,
851 --
852 --       south_rh_out    => open,
853 --       south_ri_out    => open,
854 --       south_re_out    => open,
855 --       south_ack_out   => '0',
856 --       south_data_out  => open,
857 --
858 --       east_rh_out     => r7_east_rh,
859 --       east_ri_out     => r7_east_ri,
860 --       east_re_out     => r7_east_re,
861 --       east_ack_out    => r7_east_ack,
862 --       east_data_out   => r7_east_data,
863 --
864 --       local_rh_out    => r7_rh_out,
865 --       local_ri_out    => r7_ri_out,
866 --       local_re_out    => r7_re_out,
867 --       local_ack_out   => r7_ack_out,
868 --       local_data_out  => r7_data_out
869 --     );
870 --
871 --    r8 : be_router
872 --     port map(
873 --       reset          => reset,
874 --
875 --       north_rh_in    => r5_south_rh,
876 --       north_ri_in    => r5_south_ri,
877 --       north_re_in    => r5_south_re,
878 --       north_ack_in   => r5_south_ack,
879 --       north_data_in  => r5_south_data,
880 --
881 --       west_rh_in     => r7_east_rh,
882 --       west_ri_in     => r7_east_ri,
883 --       west_re_in     => r7_east_re,
884 --       west_ack_in    => r7_east_ack,
885 --       west_data_in   => r7_east_data,
886 --
887 --       south_rh_in    => '0',
888 --       south_ri_in    => '0',
889 --       south_re_in    => '0',
890 --       south_ack_in   => open,
891 --       south_data_in  => FLIT_ZERO,
892 --
893 --       east_rh_in     => '0',
894 --       east_ri_in     => '0',
895 --       east_re_in     => '0',
896 --       east_ack_in    => open,
897 --       east_data_in   => FLIT_ZERO,
898 --
899 --       local_rh_in    => r8_rh_in,
900 --       local_ri_in    => r8_ri_in,
```

```
901 --       local_re_in   => r8_re_in,
902 --       local_ack_in  => r8_ack_in,
903 --       local_data_in => r8_data_in,
904 --
905 --       -- Output ports --
906 --       north_rh_out     => r8_north_rh,
907 --       north_ri_out     => r8_north_ri,
908 --       north_re_out     => r8_north_re,
909 --       north_ack_out    => r8_north_ack,
910 --       north_data_out   => r8_north_data,
911 --
912 --       west_rh_out      => r8_west_rh,
913 --       west_ri_out      => r8_west_ri,
914 --       west_re_out      => r8_west_re,
915 --       west_ack_out     => r8_west_ack,
916 --       west_data_out    => r8_west_data,
917 --
918 --       south_rh_out     => open,
919 --       south_ri_out     => open,
920 --       south_re_out     => open,
921 --       south_ack_out    => '0',
922 --       south_data_out   => open,
923 --
924 --       east_rh_out      => open,
925 --       east_ri_out      => open,
926 --       east_re_out      => open,
927 --       east_ack_out     => '0',
928 --       east_data_out    => open,
929 --
930 --       local_rh_out     => r8_rh_out,
931 --       local_ri_out     => r8_ri_out,
932 --       local_re_out     => r8_re_out,
933 --       local_ack_out    => r8_ack_out,
934 --       local_data_out   => r8_data_out
935 --     );
936
937 end architecture;
```

A.5.5.3 or1200_ocp.vhd

```
 1 --From:
 2 --  Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart.
 3 --  A noc-based soc executing a ray tracer, using synchronous multiprocessing
       ,
 4 --  2005. IMM, DTU. Polyteknisk Midtvejs Projekt.
 5
 6 library IEEE;
 7 use IEEE.std_logic_1164.all;
 8 use IEEE.std_logic_arith.all;
 9 use work.types.all;
10
11 entity or1200_ocp is
12
13   port (
14     data_i : out std_logic_vector(76 downto 0);  -- Debug output
15     data_d : out std_logic_vector(76 downto 0);  -- Debug output
16     clk_i            : in  std_logic;      -- Clock
17     rst_i            : in  std_logic;      -- Reset
18 -- OCP Master interface signals
19     ocp_MCmd_o       : out MCmdEncoding;   -- OCP master command
20     ocp_Maddr_o      : out std_logic_vector(addr_width-1 downto 0);
21                                            -- OCP master address
22     ocp_MData_o      : out std_logic_vector(data_width-1 downto 0);
```

```
23                                          -- OCP master data
24      ocp_MByteEn_o    : out std_logic_vector(3 downto 0);  -- OCP master byte
            enable
25      ocp_SCmdAccept_i : in  std_logic;      -- OCP slave command accept
26      ocp_SResp_i      : in  SRespEncoding;  -- OCP slave response
27      ocp_SData_i      : in  std_logic_vector(data_width-1 downto 0));
28  end or1200_ocp;
29
30  architecture arch of or1200_ocp is
31
32    component or1200_top
33      generic(
34        dw          :     integer := 32;
35        aw          :     integer := 32;
36        ppic_ints   :     integer := 20
37        );
38      port(
39        clk_i       : in  std_logic;
40        rst_i       : in  std_logic;
41        pic_ints_i  : in  std_logic_vector(ppic_ints-1 downto 0);
42        clmode_i    : in  std_logic_vector(1 downto 0);
43  -- Instruction WISHBONE interface
44        iwb_clk_i   : in  std_logic;
45        iwb_rst_i   : in  std_logic;
46        iwb_ack_i   : in  std_logic;
47        iwb_err_i   : in  std_logic;
48        iwb_rty_i   : in  std_logic;
49        iwb_dat_i   : in  std_logic_vector(31 downto 0);
50        iwb_cyc_o   : out std_logic;
51        iwb_adr_o   : out std_logic_vector(31 downto 0);
52        iwb_stb_o   : out std_logic;
53        iwb_we_o    : out std_logic;
54        iwb_sel_o   : out std_logic_vector(3 downto 0);
55        iwb_dat_o   : out std_logic_vector(31 downto 0);
56        iwb_cab_o   : out std_logic;
57
58  -- Data WISHBONE interface
59        dwb_clk_i : in  std_logic;
60        dwb_rst_i : in  std_logic;
61        dwb_ack_i : in  std_logic;
62        dwb_err_i : in  std_logic;
63        dwb_rty_i : in  std_logic;
64        dwb_dat_i : in  std_logic_vector(31 downto 0);
65        dwb_cyc_o : out std_logic;
66        dwb_adr_o : out std_logic_vector(31 downto 0);
67        dwb_stb_o : out std_logic;
68        dwb_we_o  : out std_logic;
69        dwb_sel_o : out std_logic_vector(3 downto 0);
70        dwb_dat_o : out std_logic_vector(31 downto 0);
71        dwb_cab_o : out std_logic;
72
73  -- Debug interface
74        dbg_stall_i : in  std_logic;
75        dbg_ewt_i   : in  std_logic;
76        dbg_lss_o   : out std_logic_vector(3 downto 0);
77        dbg_is_o    : out std_logic_vector(1 downto 0);
78        dbg_wp_o    : out std_logic_vector(10 downto 0);
79        dbg_bp_o    : out std_logic;
80        dbg_stb_i   : in  std_logic;
81        dbg_we_i    : in  std_logic;
82        dbg_adr_i   : in  std_logic_vector(31 downto 0);
83        dbg_dat_i   : in  std_logic_vector(31 downto 0);
84        dbg_dat_o   : out std_logic_vector(31 downto 0);
85        dbg_ack_o   : out std_logic;
86
```

```
87  -- Power Management interface
88        pm_cpustall_i  : in  std_logic;
89        pm_clksd_o     : out std_logic_vector(3 downto 0);
90        pm_dc_gate_o   : out std_logic;
91        pm_ic_gate_o   : out std_logic;
92        pm_dmmu_gate_o : out std_logic;
93        pm_immu_gate_o : out std_logic;
94        pm_tt_gate_o   : out std_logic;
95        pm_cpu_gate_o  : out std_logic;
96        pm_wakeup_o    : out std_logic;
97        pm_lvolt_o     : out std_logic
98        );
99    end component;
100
101   component or1200_mem_if
102
103   port (
104     clk_i : in std_logic;               -- Clock input
105     rst_i : in std_logic;               -- Reset input
106
107     -- WISHBONE Master interface signals
108     iwb_clk_o  : out std_logic;          -- WISHBONE clock
109     iwb_rst_o  : out std_logic;          -- WISHBONE reset
110     iwb_ack_o  : out std_logic;          -- WISHBONE Acknowledge
111     iwb_err_o  : out std_logic;          -- WISHBONE error
112     iwb_rty_o  : out std_logic;          -- WISHBONE retry
113     iwb_dat_o  : out std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data
114     iwb_cyc_i  : in  std_logic;          -- WISHBONE cycle
115     iwb_adr_i  : in  std_logic_vector(addr_width-1 downto 0);  -- WISHBONE
            address
116     iwb_stb_i  : in  std_logic;          -- WISHBONE stb
117     iwb_we_i   : in  std_logic;          -- WISHBONE write-enable
118     iwb_sel_i : in  std_logic_vector(3 downto 0);  -- WISHBONE select
119     iwb_dat_i  : in  std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data in
120     iwb_cab_i  : in  std_logic;          -- WISHBONE cab
121
122         -- WISHBONE Master interface signals
123     dwb_clk_o  : out std_logic;          -- WISHBONE clock
124     dwb_rst_o  : out std_logic;          -- WISHBONE reset
125     dwb_ack_o  : out std_logic;          -- WISHBONE Acknowledge
126     dwb_err_o  : out std_logic;          -- WISHBONE error
127     dwb_rty_o  : out std_logic;          -- WISHBONE retry
128     dwb_dat_o  : out std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data
129     dwb_cyc_i  : in  std_logic;          -- WISHBONE cycle
130     dwb_adr_i  : in  std_logic_vector(addr_width-1 downto 0);  -- WISHBONE
            address
131     dwb_stb_i  : in  std_logic;          -- WISHBONE stb
132     dwb_we_i   : in  std_logic;          -- WISHBONE write-enable
133     dwb_sel_i : in  std_logic_vector(3 downto 0);  -- WISHBONE select
134     dwb_dat_i  : in  std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data in
135     dwb_cab_i  : in  std_logic;          -- WISHBONE cab
136
137     -- OCP Master interface signals
138     ocp_MCmd_o       : out MCmdEncoding;  -- OCP master command
139     ocp_Maddr_o      : out std_logic_vector(addr_width-1 downto 0);
140                                          -- OCP master address
141     ocp_MData_o      : out std_logic_vector(data_width-1 downto 0);
142                                          -- OCP master data
143     ocp_MByteEn_o    : out std_logic_vector(3 downto 0);  -- OCP master byte
            enable
144     ocp_SCmdAccept_i : in  std_logic;   -- OCP slave command accept
```

```
145     ocp_SResp_i      : in  SRespEncoding;  -- OCP slave response
146     ocp_SData_i      : in  std_logic_vector(data_width-1 downto 0));  -- OCP
            slave data
147
148 end component;
149
150
151   signal reset_inv : std_logic;  -- Inverted clock
152   signal zero32 : std_logic_vector(31 downto 0);
153   signal pic_ints : std_logic_vector(19 downto 0);
154   signal clmode : std_logic_vector(1 downto 0);
155   signal iwb_clk, iwb_rst, iwb_ack, iwb_err, dwb_clk, dwb_rst, dwb_ack,
          dwb_err, iwb_rty, dwb_rty, iwb_cyc, iwb_stb, iwb_we, iwb_cab, dwb_cyc,
           dwb_stb, dwb_we, dwb_cab : std_logic;
156   signal iwb_sel, dwb_sel : std_logic_vector(3 downto 0);
157   signal iwb_dati, iwb_dato, dwb_dati, dwb_dato, dwb_adr, iwb_adr :
          std_logic_vector(31 downto 0);
158
159 begin  -- arch
160
161 data_i <= iwb_ack & iwb_dati & iwb_cyc & iwb_adr & iwb_we & iwb_sel & "000000
        ";
162 data_d <= dwb_ack & dwb_dato & dwb_cyc & dwb_adr & dwb_we & dwb_sel & "000000
        ";
163
164   reset_inv <= not rst_i;
165   zero32    <= X"00000000";
166   pic_ints  <= "00000000000000000000";
167   clmode    <= "00";                      -- Same clock for WISHBONE and CPU
168
169   theCPU : or1200_top
170     port map(
171       clk_i      => clk_i,
172       rst_i      => reset_inv,
173       pic_ints_i => pic_ints,
174       clmode_i   => clmode,
175 -- Instruction WISHBONE interface
176       iwb_clk_i  => clk_i,
177       iwb_rst_i  => reset_inv,
178       iwb_ack_i  => iwb_ack,
179       iwb_err_i  => iwb_err,
180       iwb_rty_i  => iwb_rty,
181       iwb_dat_i  => iwb_dati,
182       iwb_cyc_o  => iwb_cyc,
183       iwb_adr_o  => iwb_adr,
184       iwb_stb_o  => iwb_stb,
185       iwb_we_o   => iwb_we,
186       iwb_sel_o  => iwb_sel,
187       iwb_dat_o  => iwb_dato,
188       iwb_cab_o  => iwb_cab,
189
190 -- Data WISHBONE interface
191       dwb_clk_i => clk_i,
192       dwb_rst_i => reset_inv,
193       dwb_ack_i => dwb_ack,
194       dwb_err_i => dwb_err,
195       dwb_rty_i => dwb_rty,
196       dwb_dat_i => dwb_dati,
197       dwb_cyc_o => dwb_cyc,
198       dwb_adr_o => dwb_adr,
199       dwb_stb_o => dwb_stb,
200       dwb_we_o  => dwb_we,
201       dwb_sel_o => dwb_sel,
202       dwb_dat_o => dwb_dato,
203       dwb_cab_o => dwb_cab,
```

```
204
205  -- Debug interface
206        dbg_stall_i => '0',
207        dbg_ewt_i   => '0',
208        dbg_lss_o   => open,
209        dbg_is_o    => open,
210        dbg_wp_o    => open,
211        dbg_bp_o    => open,
212        dbg_stb_i   => '0',
213        dbg_we_i    => '0',
214        dbg_adr_i   => zero32,
215        dbg_dat_i   => zero32,
216        dbg_dat_o   => open,
217        dbg_ack_o   => open,
218
219  -- Power Management interface
220        pm_cpustall_i  => '0',
221        pm_clksd_o     => open,
222        pm_dc_gate_o   => open,
223        pm_ic_gate_o   => open,
224        pm_dmmu_gate_o => open,
225        pm_immu_gate_o => open,
226        pm_tt_gate_o   => open,
227        pm_cpu_gate_o  => open,
228        pm_wakeup_o    => open,
229        pm_lvolt_o     => open
230        );
231
232  ocpif : or1200_mem_if
233
234    port map (
235      clk_i => clk_i,
236      rst_i => rst_i,
237
238      -- WISHBONE Master interface signals
239      iwb_clk_o  => iwb_clk,
240      iwb_rst_o  => iwb_rst,
241      iwb_ack_o  => iwb_ack,
242      iwb_err_o  => iwb_err,
243      iwb_rty_o  => iwb_rty,
244      iwb_dat_o  => iwb_dati,
245      iwb_cyc_i  => iwb_cyc,
246      iwb_adr_i  => iwb_adr,
247      iwb_stb_i  => iwb_stb,
248      iwb_we_i   => iwb_we,
249      iwb_sel_i  => iwb_sel,
250      iwb_dat_i  => iwb_dato,
251      iwb_cab_i  => iwb_cab,
252
253          -- WISHBONE Master interface signals
254      dwb_clk_o  => dwb_clk,
255      dwb_rst_o  => dwb_rst,
256      dwb_ack_o  => dwb_ack,
257      dwb_err_o  => dwb_err,
258      dwb_rty_o  => dwb_rty,
259      dwb_dat_o  => dwb_dati,
260      dwb_cyc_i  => dwb_cyc,
261      dwb_adr_i  => dwb_adr,
262      dwb_stb_i  => dwb_stb,
263      dwb_we_i   => dwb_we,
264      dwb_sel_i  => dwb_sel,
265      dwb_dat_i  => dwb_dato,
266      dwb_cab_i  => dwb_cab,
267
268      -- OCP Master interface signals
```

```
269     ocp_MCmd_o       => ocp_MCmd_o,
270     ocp_Maddr_o      => ocp_Maddr_o,
271     ocp_MData_o      => ocp_MData_o,
272     ocp_MByteEn_o    => ocp_MByteEn_o,
273     ocp_SCmdAccept_i => ocp_SCmdAccept_i,
274     ocp_SResp_i      => ocp_SResp_i,
275     ocp_SData_i      => ocp_SData_i);
276
277
278 end arch;
```

A.5.5.4 or1200_mem_if.vhd

```
1   --From:
2   --  Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart.
3   --  A noc-based soc executing a ray tracer, using synchronous multiprocessing
        ,
4   --  2005. IMM, DTU. Polyteknisk Midtvejs Projekt.
5
6   library IEEE;
7   use IEEE.std_logic_1164.all;
8   use work.types.all;
9
10  entity or1200_mem_if is
11
12    port (
13      clk_i : in std_logic;               -- Clock input
14      rst_i : in std_logic;               -- Reset input
15
16      -- WISHBONE Master interface signals
17      iwb_clk_o  : out std_logic;          -- WISHBONE clock
18      iwb_rst_o  : out std_logic;          -- WISHBONE reset
19      iwb_ack_o  : out std_logic;          -- WISHBONE Acknowledge
20      iwb_err_o  : out std_logic;          -- WISHBONE error
21      iwb_rty_o  : out std_logic;          -- WISHBONE retry
22      iwb_dat_o  : out std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data
23      iwb_cyc_i  : in  std_logic;          -- WISHBONE cycle
24      iwb_adr_i  : in  std_logic_vector(addr_width-1 downto 0);  -- WISHBONE
            address
25      iwb_stb_i  : in  std_logic;          -- WISHBONE stb
26      iwb_we_i   : in  std_logic;          -- WISHBONE write-enable
27      iwb_sel_i : in  std_logic_vector(3 downto 0);  -- WISHBONE select
28      iwb_dat_i  : in  std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data in
29      iwb_cab_i  : in  std_logic;          -- WISHBONE cab
30
31          -- WISHBONE Master interface signals
32      dwb_clk_o  : out std_logic;          -- WISHBONE clock
33      dwb_rst_o  : out std_logic;          -- WISHBONE reset
34      dwb_ack_o  : out std_logic;          -- WISHBONE Acknowledge
35      dwb_err_o  : out std_logic;          -- WISHBONE error
36      dwb_rty_o  : out std_logic;          -- WISHBONE retry
37      dwb_dat_o  : out std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data
38      dwb_cyc_i  : in  std_logic;          -- WISHBONE cycle
39      dwb_adr_i  : in  std_logic_vector(addr_width-1 downto 0);  -- WISHBONE
            address
40      dwb_stb_i  : in  std_logic;          -- WISHBONE stb
41      dwb_we_i   : in  std_logic;          -- WISHBONE write-enable
42      dwb_sel_i : in  std_logic_vector(3 downto 0);  -- WISHBONE select
43      dwb_dat_i  : in  std_logic_vector(data_width-1 downto 0);  -- WISHBONE
            data in
```

```
44      dwb_cab_i  : in  std_logic;          -- WISHBONE cab
45
46      -- OCP Master interface signals
47      ocp_MCmd_o       : out MCmdEncoding;  -- OCP master command
48      ocp_Maddr_o      : out std_logic_vector(addr_width-1 downto 0);
49                                            -- OCP master address
50      ocp_MData_o      : out std_logic_vector(data_width-1 downto 0);
51                                            -- OCP master data
52      ocp_MByteEn_o    : out std_logic_vector(3 downto 0);  -- OCP master byte
            enable
53      ocp_SCmdAccept_i : in  std_logic;   -- OCP slave command accept
54      ocp_SResp_i      : in  SRespEncoding;  -- OCP slave response
55      ocp_SData_i      : in  std_logic_vector(data_width-1 downto 0));  -- OCP
            slave data
56
57  end or1200_mem_if;
58
59  architecture interface of or1200_mem_if is
60
61    type state is (STATE_REQUEST, STATE_SETUP_INST, STATE_SETUP_DATA,
          STATE_WAIT_INST_RD, STATE_WAIT_DATA_RD);  -- States for FSM
62  signal current_state, next_state : state;
63
64  begin  -- interface
65
66    iwb_clk_o <= clk_i;
67    dwb_clk_o <= clk_i;
68    iwb_rst_o <= not rst_i;
69    dwb_rst_o <= not rst_i;
70    iwb_err_o <= '0';
71    dwb_err_o <= '0';
72    iwb_rty_o <= '0';
73    dwb_rty_o <= '0';
74    iwb_dat_o <= ocp_SData_i;
75    dwb_dat_o <= ocp_SData_i;
76
77  -- purpose: FSM Logic
78  -- type   : combinational
79  -- inputs : iwb_cyc_i, iwb_adr_i, iwb_stb_i, iwb_we_i, iwb_sel_i, iwb_dat_i
80  -- outputs:
81  logic: process (current_state, iwb_cyc_i, iwb_adr_i, iwb_we_i, iwb_sel_i,
        iwb_dat_i, dwb_cyc_i, dwb_adr_i, dwb_we_i, dwb_sel_i, dwb_dat_i,
        ocp_SCmdAccept_i, ocp_SResp_i, ocp_SData_i)
82  begin  -- process logic
83    case current_state is
84      when STATE_REQUEST =>
85        if dwb_cyc_i = '1' then
86          next_state <= STATE_SETUP_DATA;
87        elsif iwb_cyc_i = '1' then
88          next_state <= STATE_SETUP_INST;
89        else
90          next_state <= STATE_REQUEST;
91        end if;
92        ocp_MCmd_o <= MCmd_IDLE;
93        ocp_Maddr_o <= dwb_adr_i;
94        ocp_MByteEn_o <= dwb_sel_i;
95        ocp_MData_o <= dwb_dat_i;
96        iwb_ack_o <= '0';
97        dwb_ack_o <= '0';
98      when STATE_SETUP_INST =>
99        if ocp_SCmdAccept_i = '1' then
100         if iwb_we_i = '1' then
101           next_state <= STATE_REQUEST;
102           iwb_ack_o <= '1';
103           ocp_MCmd_o <= MCmd_WR;
```

```
104           else
105             next_state <= STATE_WAIT_INST_RD;
106             iwb_ack_o <= '0';
107             ocp_MCmd_o <= MCmd_RD;
108           end if;
109         else
110           next_state <= STATE_SETUP_INST;
111           iwb_ack_o <= '0';
112           if iwb_we_i = '1' then
113             ocp_MCmd_o <= MCmd_WR;
114           else
115             ocp_MCmd_o <= MCmd_RD;
116           end if;
117         end if;
118         ocp_Maddr_o <= iwb_adr_i;
119         ocp_MByteEn_o <= iwb_sel_i;
120         ocp_MData_o <= iwb_dat_i;
121         dwb_ack_o <= '0';
122       when STATE_SETUP_DATA =>
123         if ocp_SCmdAccept_i = '1' then
124           if dwb_we_i = '1' then
125             next_state <= STATE_REQUEST;
126             dwb_ack_o <= '1';
127             ocp_MCmd_o <= MCmd_WR;
128           else
129             next_state <= STATE_WAIT_DATA_RD;
130             dwb_ack_o <= '0';
131             ocp_MCmd_o <= MCmd_RD;
132           end if;
133         else
134           next_state <= STATE_SETUP_DATA;
135           dwb_ack_o <= '0';
136           if dwb_we_i = '1' then
137             ocp_MCmd_o <= MCmd_WR;
138           else
139             ocp_MCmd_o <= MCmd_RD;
140           end if;
141         end if;
142         ocp_Maddr_o <= dwb_adr_i;
143         ocp_MByteEn_o <= dwb_sel_i;
144         ocp_MData_o <= dwb_dat_i;
145         iwb_ack_o <= '0';
146       when STATE_WAIT_INST_RD =>
147         if ocp_SResp_i = SResp_NULL then
148           next_state <= STATE_WAIT_INST_RD;
149           ocp_MCmd_o <= MCmd_IDLE;
150           ocp_Maddr_o <= iwb_adr_i;
151           ocp_MByteEn_o <= iwb_sel_i;
152           ocp_MData_o <= iwb_dat_i;
153           iwb_ack_o <= '0';
154         else
155           next_state <= STATE_REQUEST;
156           ocp_MCmd_o <= MCmd_IDLE;
157           ocp_Maddr_o <= iwb_adr_i;
158           ocp_MByteEn_o <= iwb_sel_i;
159           ocp_MData_o <= iwb_dat_i;
160           iwb_ack_o <= '1';
161         end if;
162         dwb_ack_o <= '0';
163       when STATE_WAIT_DATA_RD =>
164         if ocp_SResp_i = SResp_NULL then
165           next_state <= STATE_WAIT_DATA_RD;
166           ocp_MCmd_o <= MCmd_IDLE;
167           ocp_Maddr_o <= dwb_adr_i;
168           ocp_MByteEn_o <= dwb_sel_i;
```

```
169           ocp_MData_o <= dwb_dat_i;
170           dwb_ack_o <= '0';
171         else
172           next_state <= STATE_REQUEST;
173           ocp_MCmd_o <= MCmd_IDLE;
174           ocp_Maddr_o <= dwb_adr_i;
175           ocp_MByteEn_o <= dwb_sel_i;
176           ocp_MData_o <= dwb_dat_i;
177           dwb_ack_o <= '1';
178         end if;
179         iwb_ack_o <= '0';
180       when others =>
181           next_state <= STATE_REQUEST;
182           ocp_MCmd_o <= MCmd_IDLE;
183           ocp_Maddr_o <= dwb_adr_i;
184           ocp_MByteEn_o <= dwb_sel_i;
185           ocp_MData_o <= dwb_dat_i;
186           dwb_ack_o <= '0';
187        iwb_ack_o <= '0';
188     end case;
189   end process logic;
190
191   -- purpose: State register
192   -- type   : sequential
193   -- inputs : clk_i, rst_i
194   -- outputs:
195   State_register: process (clk_i, rst_i)
196   begin  -- process State register
197     if rst_i = '0' then                 -- asynchronous reset (active low)
198       current_state <= STATE_REQUEST;
199     elsif clk_i'event and clk_i = '1' then  -- rising clock edge
200       current_state <= next_state;
201     end if;
202   end process State_register;
203
204   end interface;
```

A.5.5.5 core_mem_ocp.vhd

```
1   --From:
2   --  Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart.
3   --  A noc-based soc executing a ray tracer, using synchronous multiprocessing
        ,
4   --  2005. IMM, DTU. Polyteknisk Midtvejs Projekt.
5
6   library IEEE;
7   use IEEE.std_logic_1164.all;
8   use IEEE.std_logic_arith.all;
9   use STD.textio.all;
10  use WORK.types.all;
11
12  entity core_mem_ocp is
13
14    port (
15      clk_i : in std_logic;               -- Clock
16      rst_i : in std_logic;               -- Reset
17      ocp_MCmd_i : in MCmdEncoding;       -- OCP master command
18      ocp_MAddr_i : in std_logic_vector(addr_width-1 downto 0);
19                                          -- OCP master address
20      ocp_MData_i : in std_logic_vector(data_width-1 downto 0); -- OCP master
            data
21      ocp_MByteEn_i : in std_logic_vector(3 downto 0);  -- OCP Master byte
            enable
```

```
22      ocp_SCmdAccept_o : out std_logic;    -- OCP slave command accept
23      ocp_SResp_o : out SRespEncoding;     -- OCP slave response
24      ocp_SData_o : out std_logic_vector(data_width-1 downto 0) -- OCP slave
            data
25      );
26  end core_mem_ocp;
27
28  architecture arch of core_mem_ocp is
29
30    component onchip_mem
31      port (
32        clka  : in  std_logic;
33        dina  : in  std_logic_VECTOR(31 downto 0);
34        addra : in  std_logic_VECTOR(12 downto 0);
35        wea   : in  std_logic_VECTOR(3 downto 0);
36        douta : out std_logic_VECTOR(31 downto 0)
37      );
38    end component;
39
40  type state is (STATE_REQUEST, STATE_RESPONSE, STATE_WAIT);
41    signal current_state, next_state : state;
42  signal mem_wea : std_logic_vector(3 downto 0);
43
44  begin  -- arch
45
46    mem : onchip_mem
47      port map (
48        clka  => clk_i,
49        dina  => ocp_MData_i,
50        addra => ocp_MAddr_i(14 downto 2),  --The memory uses word-addressing,
              the OCP byte addressing.,
51        wea   => mem_wea,
52        douta => ocp_SData_o
53      );
54
55    ocp_SCmdAccept_o <= '1';
56
57    next_state_logic: process(current_state, ocp_MCmd_i, ocp_MByteEn_i)
58    begin
59      case current_state is
60        when STATE_REQUEST =>
61          ocp_SResp_o <= SResp_NULL;
62
63          if ocp_MCmd_i = MCmd_WR then
64            -- Write
65            mem_wea <= ocp_MByteEn_i;
66            next_state <= STATE_REQUEST;
67
68          elsif ocp_MCmd_i = MCmd_RD then
69            -- Read
70            mem_wea <= "0000";
71            next_state <= STATE_RESPONSE;
72
73          else
74            mem_wea <= "0000";
75            next_state <= STATE_REQUEST;
76          end if;
77
78        when STATE_RESPONSE =>
79          mem_wea <= "0000";
80          ocp_SResp_o <= SResp_DVA;
81          next_state <= STATE_WAIT;
82        when STATE_WAIT =>
83          mem_wea <= "0000";
84          --ocp_SResp_o <= SResp_DVA;
```

```
85          ocp_SResp_o <= SResp_NULL;
86          if ocp_MCmd_i = MCmd_IDLE then
87            next_state <= STATE_REQUEST;
88          else
89            next_state <= STATE_WAIT;
90          end if;
91      end case;
92    end process;
93
94    state_register : process(clk_i, rst_i, next_state)
95    begin
96      if rst_i = '0' then
97        current_state <= STATE_REQUEST;
98      elsif rising_edge(clk_i) then
99        current_state <= next_state;
100     end if;
101   end process;
102
103 end arch;
```

A.5.5.6 semaphore.vhd

```
1   --From:
2   --  Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart.
3   --  A noc-based soc executing a ray tracer, using synchronous multiprocessing
        ,
4   --  2005. IMM, DTU. Polyteknisk Midtvejs Projekt.
5
6   library IEEE;
7   use IEEE.std_logic_1164.all;
8   use IEEE.std_logic_arith.all;
9   use work.types.all;
10
11  entity semaphore_ocp is
12    generic (
13      semaphores : integer := 5);        -- log2(Number of semaphores)
14    port (
15      clk_i : in std_logic;               -- Clock
16      rst_i : in std_logic;               -- Reset
17      -- OCP Slave interface
18      ocp_MCmd_i       : in  MCmdEncoding;   -- OCP master command
19      ocp_Maddr_i      : in  std_logic_vector(addr_width-1 downto 0);
20                                             -- OCP master address
21      ocp_MData_i      : in  std_logic_vector(data_width-1 downto 0);
22                                             -- OCP master data
23      ocp_MByteEn_i    : in std_logic_vector(3 downto 0); -- OCP master
            byteenable
24      ocp_SCmdAccept_o : out std_logic;      -- OCP slave command accept
25      ocp_SResp_o      : out SRespEncoding;  -- OCP slave response
26      ocp_SData_o      : out std_logic_vector(data_width-1 downto 0) -- OCP
            slave data
27      );
28  end semaphore_ocp;
29
30  architecture arch of semaphore_ocp is
31
32  type state is (STATE_IDLE, STATE_UPDATE);  -- States for FSM
33  signal current_state, next_state : state;
34
35  signal sem_state, sem_state_update :
36    std_logic_vector((2 ** semaphores)-1 downto 0);  -- Semaphore state
37
38  begin  -- arch
```

```
39    ocp_SCmdAccept_o <= '1';
40
41  -- purpose: Semaphore
42  -- type   : combinational
43  -- inputs : ocp_MCmd_i, ocp_Maddr_i, ocp_MData_i
44  -- outputs: ocp_SResp_o, ocp_SData_o
45  Semaphore: process (ocp_MCmd_i, ocp_Maddr_i, ocp_MData_i)
46  begin  -- process Semaphore
47    case ocp_MCmd_i is
48      when MCmd_RD =>                     -- Test and set
49        ocp_SResp_o <= SResp_DVA;
50        ocp_SData_o <= X"0000000" & "000" & sem_state(conv_integer(
51          unsigned(ocp_Maddr_i(1+semaphores downto 2))));
52        sem_state_update <= sem_state;
53        sem_state_update(conv_integer(unsigned(
54          ocp_Maddr_i(1+semaphores downto 2)))) <= '0';
55      when MCmd_WR =>                     -- Set to 1
56        ocp_SResp_o <= SResp_NULL;
57        ocp_SData_o <= X"00000000";
58        sem_state_update <= sem_state;
59        sem_state_update(conv_integer(unsigned(
60          ocp_Maddr_i(1+semaphores downto 2)))) <= '1';
61      when others =>
62        ocp_SResp_o <= SResp_NULL;
63        ocp_SData_o <= X"00000000";
64        sem_state_update <= sem_state;
65    end case;
66  end process Semaphore;
67
68  -- purpose: Semaphore state register
69  -- type   : sequential
70  -- inputs : clk_i, rst_i
71  -- outputs:
72  State_register: process (clk_i, rst_i)
73  begin  -- process State register
74    if rst_i = '0' then                   -- asynchronous reset (active low)
75      sem_state <= (others => '1');
76    elsif clk_i'event and clk_i = '1' then  -- rising clock edge
77      sem_state <= sem_state_update;
78    end if;
79  end process State_register;
80
81  end arch;
```

A.5.5.7 uart16550_ocp.vhd

```
1   --From:
2   --  Morten Sleth Rasmussen, Christian Place Pedersen, and Matthias Bo Stuart.
3   --  A noc-based soc executing a ray tracer, using synchronous multiprocessing
        ,
4   --  2005. IMM, DTU. Polyteknisk Midtvejs Projekt.
5
6   library IEEE;
7   use IEEE.std_logic_1164.all;
8   use IEEE.std_logic_arith.all;
9   use work.types.all;
10
11  entity uart16550_ocp is
12
13    port (
14      clk_i            : in  std_logic;      -- Clock
15      rst_i            : in  std_logic;      -- Reset
16      -- OCP slave interface
```

```
17      ocp_MCmd_i       : in  MCmdEncoding;   -- OCP master command
18      ocp_Maddr_i      : in  std_logic_vector(addr_width-1 downto 0);
19                                             -- OCP master address
20      ocp_MData_i      : in  std_logic_vector(data_width-1 downto 0);
21                                             -- OCP master data
22      ocp_MByteEn_i    : in std_logic_vector(3 downto 0); -- OCP master
            byteenable
23      ocp_SCmdAccept_o : out std_logic;      -- OCP slave command accept
24      ocp_SResp_o      : out SRespEncoding;  -- OCP slave response
25      ocp_SData_o      : out std_logic_vector(data_width-1 downto 0);
26      -- Interrupt
27      int_o            : out std_logic;   -- Interrupt signal
28      -- RS232 interface
29      tx               : out std_logic;   -- TX pad
30      rx               : in  std_logic;   -- RX pad
31      rts              : out std_logic;   -- RTS pad
32      cts              : in  std_logic;   -- CTS pad
33      dtr              : out std_logic;   -- DTR pad
34      dsr              : in  std_logic;   -- DSR pad
35      ri               : in  std_logic;   -- RI pad
36      dcd              : in  std_logic);  -- DCD pad
37  end uart16550_ocp;
38
39  architecture struct of uart16550_ocp is
40   component uart_top
41       generic(
42           uart_data_width : integer := 32;
43           uart_addr_width : integer := 5
44       );
45       port(
46           wb_clk_i        : in    std_logic;
47           wb_rst_i        : in    std_logic;
48           wb_adr_i        : in    std_logic_vector(4 downto 0);
49           wb_dat_i        : in    std_logic_vector(data_width-1 downto 0);
50           wb_dat_o        : out   std_logic_vector(data_width-1 downto 0);
51           wb_we_i         : in    std_logic;
52           wb_stb_i        : in    std_logic;
53           wb_cyc_i        : in    std_logic;
54           wb_ack_o        : out   std_logic;
55           wb_sel_i        : in    std_logic_vector(3 downto 0);
56           int_o           : out   std_logic;
57           stx_pad_o       : out   std_logic;
58           srx_pad_i       : in    std_logic;
59           rts_pad_o       : out   std_logic;
60           cts_pad_i       : in    std_logic;
61           dtr_pad_o       : out   std_logic;
62           dsr_pad_i       : in    std_logic;
63           ri_pad_i        : in    std_logic;
64           dcd_pad_i       : in    std_logic
65       );
66   end component;
67
68    component OCPm_to_WBm is
69    port (
70      clk_i : in std_logic;
71      rst_i : in std_logic;
72      -- OCP Master interface signals
73      ocp_MCmd_i       : in  MCmdEncoding;                                -- OCP
            master command
74      ocp_Maddr_i      : in  std_logic_vector(addr_width-1 downto 0);     --
            OCP master address
75      ocp_MData_i      : in  std_logic_vector(data_width-1 downto 0);     --
            OCP master data
76      ocp_MByteEn_i    : in  std_logic_vector(3 downto 0);              -- OCP
            master byte enable
```

```
77      ocp_SCmdAccept_o : out std_logic;                                    -- OCP
            slave command accept
78      ocp_SResp_o      : out SRespEncoding;                                -- OCP
            slave response
79      ocp_SData_o      : out  std_logic_vector(data_width-1 downto 0);     --
            OCP slave data
80
81      -- WISHBONE Master interface signals
82      wb_ack_i  : in  std_logic;                                  -- WISHBONE
            Acknowledge
83      wb_err_i  : in  std_logic;                                  -- WISHBONE
            error
84      wb_rty_i  : in  std_logic;                                  -- WISHBONE
            retry
85      wb_dat_i  : in  std_logic_vector(data_width-1 downto 0);    -- WISHBONE
            data
86      wb_cyc_o  : out std_logic;                                  -- WISHBONE
            cycle
87      wb_adr_o  : out std_logic_vector(addr_width-1 downto 0);    -- WISHBONE
            address
88      wb_stb_o  : out std_logic;                                  -- WISHBONE stb
89      wb_we_o   : out std_logic;                                  -- WISHBONE
            write-enable
90      wb_sel_o  : out std_logic_vector(3 downto 0);               -- WISHBONE
            select
91      wb_dat_o  : out std_logic_vector(data_width-1 downto 0);    -- WISHBONE
            data in
92      wb_cab_o  : out std_logic                                   -- WISHBONE cab
93    );
94    end component;
95
96  signal s_wb_dat_i, s_wb_dat_o, s_wb_addr : std_logic_vector(data_width-1
        downto 0);
97                                          -- Wishbone data in/output
98  signal s_rst, s_wb_we, s_wb_stb, s_wb_cyc, s_wb_ack : std_logic;
99                                          -- Wishbone handshake signals
100 --  signal s_wb_sel : std_logic_vector(3 downto 0);  -- Wishbone select
101
102  begin  -- struct
103
104 s_rst <= not rst_i;
105
106  uart : uart_top
107    port map (
108      wb_clk_i  => clk_i,
109      wb_rst_i  => s_rst,
110      wb_adr_i  => s_wb_addr(4 downto 0),
111      wb_dat_i  => s_wb_dat_i,
112      wb_dat_o  => s_wb_dat_o,
113      wb_we_i   => s_wb_we,
114      wb_stb_i  => s_wb_stb,
115      wb_cyc_i  => s_wb_cyc,
116      wb_ack_o  => s_wb_ack,
117      wb_sel_i  => ocp_MByteEn_i,
118     int_o     => int_o,
119     stx_pad_o => tx,
120     srx_pad_i => rx,
121     rts_pad_o => rts,
122     cts_pad_i => cts,
123     dtr_pad_o => dtr,
124     dsr_pad_i => dsr,
125     ri_pad_i  => ri,
126     dcd_pad_i => dcd);
127
128 ocp_wrapper : OCPm_to_WBm
```

```
129   port map(
130     clk_i             =>  clk_i,
131     rst_i             =>  rst_i,
132     ocp_MCmd_i        =>  ocp_MCmd_i,
133     ocp_Maddr_i       =>  ocp_Maddr_i,
134     ocp_MData_i       =>  ocp_MData_i,
135     ocp_MByteEn_i     =>  ocp_MByteEn_i,
136     ocp_SCmdAccept_o  =>  ocp_SCmdAccept_o,
137     ocp_SResp_o       =>  ocp_SResp_o,
138     ocp_SData_o       =>  ocp_SData_o,
139
140     wb_ack_i          =>  s_wb_ack,
141     wb_err_i          =>  '0',
142     wb_rty_i          =>  '0',
143     wb_dat_i          =>  s_wb_dat_o,
144     wb_cyc_o          =>  s_wb_cyc,
145     wb_adr_o          =>  s_wb_addr,
146     wb_stb_o          =>  s_wb_stb,
147     wb_we_o           =>  s_wb_we,
148     wb_sel_o          =>  open,
149     wb_dat_o          =>  s_wb_dat_i,
150     wb_cab_o          =>  open
151   );
152
153
154 --s_wb_dat_i <= ocp_MData_i;
155
156 ---- purpose: OCP slave interface
157 ---- type   : combinational
158 ---- inputs : ocp_MCmd_i, ocp_Maddr_i, ocp_MData_i
159 ---- outputs: ocp_SResp_o, ocp_SData_o
160 --Semaphore: process (ocp_MCmd_i, ocp_Maddr_i, ocp_MData_i, s_wb_dat_o,
        s_wb_ack)
161 --begin
162 --  case ocp_MCmd_i is
163 --    when MCmd_RD =>
164 --      s_wb_stb <= '1';
165 --      s_wb_cyc <= '1';
166 --      s_wb_we <= '0';
167 --      if s_wb_ack = '1' then
168 --        ocp_SResp_o <= SResp_DVA;
169 --      else
170 --        ocp_SResp_o <= SResp_NULL;
171 --      end if;
172 --      ocp_SData_o <= s_wb_dat_o;
173 --    when MCmd_WR =>
174 --      s_wb_stb <= '1';
175 --      s_wb_cyc <= '1';
176 --      s_wb_we <= '1';
177 --      ocp_SResp_o <= SResp_NULL;
178 --      ocp_SData_o <= s_wb_dat_o;
179 --    when others =>
180 --      s_wb_stb <= '0';
181 --      s_wb_cyc <= '0';
182 --      s_wb_we <= '0';
183 --      ocp_SResp_o <= SResp_NULL;
184 --      ocp_SData_o <= X"00000000";
185 --  end case;
186 --end process Semaphore;
187 --
188 --ocp_SCmdAccept_o <= s_wb_ack;
189
190 end struct;
```

A.5.5.8 types.vhd

```
1  -- Constant and types
2
3  library IEEE;
4  use IEEE.std_logic_1164.all;
5
6  package types is
7
8    --NoC
9    constant FLIT_SIZE  : integer   := 32;     -- flit size in bits
10   constant FLIT_UNDEF : std_logic_vector(FLIT_SIZE-1 downto 0) := (others =>
         'U');  -- undefined flit
11   constant FLIT_ZERO  : std_logic_vector(FLIT_SIZE-1 downto 0) := (others =>
         '0');  -- zero flit
12   subtype  flit_data is std_logic_vector(FLIT_SIZE-1 downto 0);
13
14   type source_hs_data is array (0 to 3) of flit_data;
15
16   --ROMs for traffic generators
17   constant SOURCE_ROM_SIZE : integer := 64;
18   type rom_type is array (SOURCE_ROM_SIZE-1 downto 0) of std_logic_vector(2+(
         FLIT_SIZE-1) downto 0);
19
20   --route lookup
21   constant ROUTE_LOOKUP_SIZE : integer := 16;
22   type route_lookup_table_type is array (ROUTE_LOOKUP_SIZE-1 downto 0) of
         std_logic_vector(2*FLIT_SIZE-1 downto 0); -- forward_path &
         reverse_path
23
24   -- Encoding of the MCmd
25   subtype MCmdEncoding is std_logic_vector(2 downto 0);
26
27   constant MCmd_IDLE : MCmdEncoding := "000";  -- Idle
28   constant MCmd_WR   : MCmdEncoding := "001";  -- Write
29   constant MCmd_RD   : MCmdEncoding := "010";  -- Read
30   constant MCmd_RDEX : MCmdEncoding := "011";  -- ReadEx
31   constant MCmd_RDL  : MCmdEncoding := "100";  -- ReadLinked
32   constant MCmd_WRNP : MCmdEncoding := "101";  -- WriteNonPost
33   constant MCmd_WRC  : MCmdEncoding := "110";  -- WriteConditional
34   constant MCmd_BCST : MCmdEncoding := "111";  -- Broadcast
35
36   -- Encoding of the SResp
37   subtype SRespEncoding is std_logic_vector(1 downto 0);
38
39   constant SResp_NULL : SRespEncoding := "00";  -- No Response
40   constant SResp_DVA  : SRespEncoding := "01";  -- Data Valid / accept
41   constant SResp_FAIL : SRespEncoding := "10";  -- Request failed
42   constant SResp_ERR  : SRespEncoding := "11";  -- Response error
43
44   -- SoC constants
45   constant addr_width     : integer := 32;  -- Width of address
46   constant data_width     : integer := 32;  -- Width of data
47   constant WORD_ZERO      : std_logic_vector(data_width-1 downto 0) := (
         others => '0');  -- Zero word
48
49 end types;
50
51 package body types is
52
53 end types;
```

A.5.5.9 route_lookup_tables.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.types.all;
4
5  package route_lookup_tables is
6
7  -- 9 router mesh
8    constant cpu0_routing_table : route_lookup_table_type := (
9      4       => x"59000000" & x"f2000000", -- (sem)
10     8       => x"58000000" & x"f2000000", -- (uart)
11     others  => x"5c000000" & x"f4000000"  -- (mem0)
12   );
13
14   constant cpu1_routing_table : route_lookup_table_type := (
15     4       => x"64000000" & x"c8000000", -- (sem)
16     8       => x"60000000" & x"C8000000", -- (uart)
17     others  => x"70000000" & x"d0000000"  -- (mem0)
18   );
19
20   constant cpu2_routing_table : route_lookup_table_type := (
21     4       => x"54000000" & x"f4000000", -- (sem)
22     8       => x"5c000000" & x"f4000000", -- (uart)
23     others  => x"52000000" & x"f8000000"  -- (mem0)
24   );
25
26   constant cpu3_routing_table : route_lookup_table_type := (
27     4       => x"50000000" & x"D0000000", -- (sem)
28     8       => x"70000000" & x"d0000000", -- (uart)
29     others  => x"48000000" & x"E0000000"  -- (mem0)
30   );
31
32   constant cpu4_routing_table : route_lookup_table_type := (
33     4       => x"44000000" & x"e0000000", -- (sem)
34     8       => x"48000000" & x"e0000000", -- (uart)
35     others  => x"42000000" & x"E8000000"  -- (mem0)
36   );
37 end route_lookup_tables;
38
39 package body route_lookup_tables is
40
41
42
43 end route_lookup_tables;
```

A.5.5.10 MPSoC_noc.ucf

```
1  NET "clk0_i" TNM_NET = "clk0_i";
2  NET "clk0_i" LOC = AH15;
3  NET "clk0_i" IOSTANDARD = "LVCMOS33";
4  TIMESPEC "TS_clk0_i" = PERIOD "clk0_i" 10000 ps;
5
6  NET "clk1_i" TNM_NET = "clk1_i";
7  NET "clk1_i" LOC = AH17;
8  NET "clk1_i" IOSTANDARD = "LVCMOS33";
9  TIMESPEC "TS_1clk_i" = PERIOD "clk1_i" 30303 ps;
10
11 NET "reset_i" LOC = E9;
12 NET "reset_i" IOSTANDARD = "LVCMOS33";
13 NET "reset_i" PULLUP;
14 NET "rx" LOC = AG15;
15 NET "rx" IOSTANDARD = "LVCMOS33";
```

```
16 NET "tx" LOC = AG20;
17 NET "tx" IOSTANDARD = "LVCMOS33";
18 NET "running" LOC = H18;
19 NET "running" IOSTANDARD = "LVCMOS25";
20 NET "running" SLEW = SLOW;
21 NET "running" PULLDOWN;
22
23 NET "*transmit_req" TIG;
24 NET "*receive_ack" TIG;
25 NET "mesh/*" TIG;
26 NET "reset" TIG;
```

A.5.6 Simulation Components

A.5.6.1 source_handshake_iterative.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity source_handshake_iterative is
8    generic(
9      starttime     : time := 50 ns;       -- starttime in ns
10     between_flits : time := 50 ns;
11     h_data        : source_hs_data := (FLIT_ZERO,FLIT_ZERO,FLIT_ZERO,
           FLIT_ZERO);
12     i_data        : source_hs_data := (FLIT_ZERO,FLIT_ZERO,FLIT_ZERO,
           FLIT_ZERO);
13     e_data        : source_hs_data := (FLIT_ZERO,FLIT_ZERO,FLIT_ZERO,
           FLIT_ZERO)
14   );
15   port (
16     rh    : out std_logic;
17     ri    : out std_logic;
18     re    : out std_logic;
19     ack   : in  std_logic;
20     data  : out flit_data
21   );
22 end source_handshake_iterative;
23
24 architecture Behavioral of source_handshake_iterative is
25   signal rh_int, ri_int, re_int : std_logic;
26   signal h_data_set, i_data_set, e_data_set : std_logic;
27   signal ii : natural := 0;
28 begin
29
30   rh <= rh_int;
31   ri <= ri_int;
32   re <= re_int;
33
34   req_h: process
35     variable i : natural := 0;
36   begin
37     rh_int <= '0';
38     h_data_set <= '0';
39     wait for starttime;
40     --while i <= 3 loop
41       if i /= 0 then
42         wait for between_flits;
```

```
43        end if;
44        rh_int <= '1';
45        h_data_set <= '1';
46        wait until ack = '1';
47        wait for 5 ns;
48        rh_int <= '0';
49        wait until ack = '0';
50        h_data_set <= '0';
51        wait until re_int = '1';  -- Wait until the complete handshake for the
              entire packet has finished.
52        wait until re_int = '0';
53        wait until ack = '0';
54        i := i+1;
55        ii <= i;
56      --end loop;
57      wait;
58    end process;
59
60    req_i: process
61    begin
62      ri_int <= '0';
63      i_data_set <= '0';
64      loop
65        wait until rh_int = '1';
66        wait until ack = '1';
67        wait until ack = '0';
68        wait for 5 ns;
69        ri_int <= '1';
70        i_data_set <= '1';
71        wait until ack = '1';
72        wait for 1 ns;
73        ri_int <= '0';
74        wait until ack = '0';
75        i_data_set <= '0';
76        wait until re_int = '1';  -- Wait until the complete handshake for the
              entire packet has finished.
77        wait until re_int = '0';
78        wait until ack = '0';
79      end loop;
80    end process;
81
82    req_e: process
83    begin
84      re_int <= '0';
85      e_data_set <= '0';
86      loop
87        wait until ri_int = '1';
88        wait until ack = '1';
89        wait until ack = '0';
90        wait for 1 ps;
91        re_int <= '1';
92        e_data_set <= '1';
93        wait until ack = '1';
94        wait for 5 ns;
95        re_int <= '0';
96        wait until ack = '0';
97        e_data_set <= '0';
98      end loop;
99    end process;
100
101   data_proc : process(h_data_set,e_data_set,i_data_set,ii)
102   begin
103     if ii > 3 then  --stop
104       data <= x"aaaaaaaa";--FLIT_ZERO;
105     elsif h_data_set = '1' then
```

```
106         data <= h_data(ii);
107       elsif i_data_set = '1' then
108         data <= i_data(ii);
109       elsif e_data_set = '1' then
110         data <= e_data(ii);
111       else
112         data <= x"aaaaaaaa";--FLIT_ZERO;
113       end if;
114     end process;
115
116   end Behavioral;
```

A.5.6.2 sink_handshake.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
7   entity sink_handshake is
8     port (
9       rh    : in  std_logic;
10      ri    : in  std_logic;
11      re    : in  std_logic;
12      ack   : out std_logic;
13      data  : in  flit_data
14    );
15  end sink_handshake;
16
17  architecture Behavioral of sink_handshake is
18    signal req : std_logic;
19
20  begin
21
22    req <= rh or ri or re;
23
24    handshake: process
25    begin
26      ack <= '0';
27      loop
28        wait until req = '1';
29        wait for 5 ns;
30        ack <= '1';
31        wait until req = '0';
32        wait for 5 ns;
33        ack <= '0';
34      end loop;
35      --wait;
36    end process;
37
38  end Behavioral;
```

A.5.6.3 ocp_master_source.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use work.types.all;
6
```

```
 7  entity ocp_master_source is
 8    port(
 9      clk_i             : in std_logic;
10      reset_i           : in std_logic;
11
12      ocp_MCmd_o        : out MCmdEncoding;
13      ocp_Maddr_o       : out std_logic_vector(addr_width-1 downto 0);
14      ocp_MData_o       : out std_logic_vector(addr_width-1 downto 0);
15      ocp_MByteEn_o     : out std_logic_vector(3 downto 0);
16      ocp_SCmdAccept_i  : in  std_logic;
17      ocp_SResp_i       : in SRespEncoding;
18      ocp_SData_i       : in std_logic_vector(addr_width-1 downto 0)
19    );
20  end ocp_master_source;
21
22  architecture Behavioral of ocp_master_source is
23
24  begin
25
26    tb : process
27    begin
28
29      ocp_MCmd_o <= (others => '0');
30      ocp_Maddr_o <= (others => '0');
31      ocp_MData_o <= (others => '0');
32      ocp_MByteEn_o <= (others => '0');
33
34      -- Wait 100 ns for global reset to finish
35      wait for 100 ns;
36      wait until clk_i = '1';
37
38      -- Write request
39      ocp_MCmd_o <= MCmd_WR;
40      ocp_Maddr_o <= x"11111111";
41      ocp_MData_o <= x"22222222";
42      ocp_MByteEn_o <= "1111";
43
44      wait until ocp_SCmdAccept_i = '1';
45      wait until clk_i = '1';
46      ocp_MCmd_o    <= MCmd_IDLE;
47      ocp_Maddr_o   <= (others => '0');
48      ocp_MData_o   <= (others => '0');
49      ocp_MByteEn_o <= (others => '0');
50
51      -- Read request
52      wait until clk_i = '0';
53      wait until clk_i = '1';
54      ocp_MCmd_o <= MCmd_RD;
55      ocp_Maddr_o <= x"44444444";
56      ocp_MByteEn_o <= "1111";
57      wait until ocp_SCmdAccept_i = '1';
58      wait until clk_i = '1';
59      ocp_MCmd_o <= MCmd_IDLE;
60      ocp_Maddr_o <= (others => '0');
61      ocp_MByteEn_o <= (others => '0');
62      wait until ocp_SResp_i = SResp_DVA;
63
64      wait; -- will wait forever
65    end process;
66
67  end Behavioral;
```

A.5.6.4 ocp_master_sink.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.types.all;
6
7  entity ocp_master_sink is
8    port(
9      clk_i             : in std_logic;
10     reset_i           : in std_logic;
11
12     ocp_MCmd_i        : in  MCmdEncoding;
13     ocp_Maddr_i       : in  std_logic_vector(addr_width-1 downto 0);
14     ocp_MData_i       : in  std_logic_vector(addr_width-1 downto 0);
15     ocp_MByteEn_i     : in  std_logic_vector(3 downto 0);
16     ocp_SCmdAccept_o  : out std_logic;
17     ocp_SResp_o       : out SRespEncoding;
18     ocp_SData_o       : out std_logic_vector(addr_width-1 downto 0)
19   );
20 end ocp_master_sink;
21
22 architecture Behavioral of ocp_master_sink is
23
24 begin
25
26   tb : process
27   begin
28     ocp_SCmdAccept_o  <=  '0';
29     ocp_SResp_o       <= (others => '0');
30     ocp_SData_o       <= (others => '0');
31
32     -- Wait 100 ns for global reset to finish
33     wait for 100 ns;
34
35     wait until ocp_MCmd_i = MCmd_WR;
36     ocp_SCmdAccept_o <= '1';
37
38     wait until clk_i = '0';
39     wait until clk_i = '1';
40
41     ocp_SCmdAccept_o <= '1';
42
43     wait until ocp_MCmd_i = MCmd_RD;
44
45     wait until clk_i = '0';
46     wait until clk_i = '1';
47
48     ocp_SCmdAccept_o <= '1';
49
50     wait until clk_i = '0';
51     wait until clk_i = '1';
52
53     ocp_SCmdAccept_o <= '0';
54
55     wait until clk_i = '0';
56     wait until clk_i = '1';
57
58     ocp_SResp_o <= SResp_DVA;
59     ocp_SData_o <= x"88888888";
60
61     wait until clk_i = '0';
62     wait until clk_i = '1';
63
```

```
64      ocp_SResp_o <= SResp_NULL;
65      ocp_SData_o <= x"00000000";
66
67      wait; -- will wait forever
68    end process;
69
70  end Behavioral;
```

A.6 C-Code

This appendix contains the original C source code from [22].

A.6.1 uart5cpu.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  volatile int *jobAlloc;
59  volatile int *uartAlloc;
60
61  volatile int nextJob = 0;
62
63  void print0();
64  void print1();
65  void print2();
66  void print3();
67  void print4();
68
69  int main(int argc, char* argv[]) {
70    jobAlloc =  SEMAPHORE_ADDRESS(1);
71    uartAlloc = SEMAPHORE_ADDRESS(2);
72
73    PASS_SEMAPHORE(jobAlloc);
74    switch(nextJob) {
75    case 0:
76      uart_init();
77      ++nextJob;
78      RELEASE_SEMAPHORE(jobAlloc);
79      print0();
80      break;
81    case 1:
82      ++nextJob;
83      RELEASE_SEMAPHORE(jobAlloc);
84      print1();
85      break;
86    case 2:
87      ++nextJob;
88      RELEASE_SEMAPHORE(jobAlloc);
89      print2();
90      break;
91    case 3:
92      ++nextJob;
93      RELEASE_SEMAPHORE(jobAlloc);
94      print3();
95      break;
96    case 4:
97      ++nextJob;
98      RELEASE_SEMAPHORE(jobAlloc);
99      print4();
100     break;
101   default:
102     RELEASE_SEMAPHORE(jobAlloc);
103     while(1) {}
104     break;
105   }
106 }
```

```
107
108 void print0() {
109   while(1) {
110     PASS_SEMAPHORE(uartAlloc);
111     uart_putc('H');
112     uart_putc('e');
113     uart_putc('l');
114     uart_putc('l');
115     uart_putc('o');
116     uart_putc('0');
117     uart_putc('\n');
118     RELEASE_SEMAPHORE(uartAlloc);
119   }
120 }
121
122 void print1() {
123   while(1) {
124     PASS_SEMAPHORE(uartAlloc);
125     uart_putc('H');
126     uart_putc('e');
127     uart_putc('l');
128     uart_putc('l');
129     uart_putc('o');
130     uart_putc('1');
131     uart_putc('\n');
132     RELEASE_SEMAPHORE(uartAlloc);
133   }
134 }
135
136 void print2() {
137   while(1) {
138     PASS_SEMAPHORE(uartAlloc);
139     uart_putc('H');
140     uart_putc('e');
141     uart_putc('l');
142     uart_putc('l');
143     uart_putc('o');
144     uart_putc('2');
145     uart_putc('\n');
146     RELEASE_SEMAPHORE(uartAlloc);
147   }
148 }
149
150 void print3() {
151   while(1) {
152     PASS_SEMAPHORE(uartAlloc);
153     uart_putc('H');
154     uart_putc('e');
155     uart_putc('l');
156     uart_putc('l');
157     uart_putc('o');
158     uart_putc('3');
159     uart_putc('\n');
160     RELEASE_SEMAPHORE(uartAlloc);
161   }
162 }
163
164 void print4() {
165   while(1) {
166     PASS_SEMAPHORE(uartAlloc);
167     uart_putc('H');
168     uart_putc('e');
169     uart_putc('l');
170     uart_putc('l');
171     uart_putc('o');
```

```
172     uart_putc('4');
173     uart_putc('\n');
174     RELEASE_SEMAPHORE(uartAlloc);
175   }
176 }
177
178 /*
179 or32-uclinux-gcc -g -c -o uart5cpu.o uart5cpu.c -I. -O2
180 or32-uclinux-ld -Tram.ld -o uart5cpu.or32 reset.o uart5cpu.o
181 or32-uclinux-nm uart5cpu.or32 | grep -v '\(compiled\)\|\(\.o$$\)\|\( [aUw] \)
        \|\(\.\.ng$$\)\|\(LASH[RL]DI\)' | sort > System.map
182 cp System.map System.map.uart5cpu
183 or32-uclinux-objcopy -O binary uart5cpu.or32  uart5cpu.bin
184 hexdump -v -e '4/1 "%02x" "\n"' uart5cpu.bin > uart5cpu.hex
185 */
```

A.6.2 board.h

```
1  #define IN_CLK          40000000 /* 10000000 */
2  #define STACK_SIZE      0x2000
3  #define UART_BAUD_RATE  115200 /* 9600 */
4  #define UART_BASE       0x80000000
5
6  /* Register access macros */
7  #define REG8(add) *((volatile unsigned char *)(add))
8  #define REG16(add) *((volatile unsigned short *)(add))
9  #define REG32(add) *((volatile unsigned long *)(add))
```

A.6.3 uart.h

```
1  #ifndef _UART_H_
2  #define _UART_H_
3
4  #define UART_RX   0 /* In:  Receive buffer (DLAB=0) */
5  #define UART_TX   0 /* Out: Transmit buffer (DLAB=0) */
6  #define UART_DLL  0 /* Out: Divisor Latch Low (DLAB=1) */
7  #define UART_DLM  1 /* Out: Divisor Latch High (DLAB=1) */
8  #define UART_IER  1 /* Out: Interrupt Enable Register */
9  #define UART_IIR  2 /* In:  Interrupt ID Register */
10 #define UART_FCR  2 /* Out: FIFO Control Register */
11 #define UART_EFR  2 /* I/O: Extended Features Register */
12         /* (DLAB=1, 16C660 only) */
13 #define UART_LCR  3 /* Out: Line Control Register */
14 #define UART_MCR  4 /* Out: Modem Control Register */
15 #define UART_LSR  5 /* In:  Line Status Register */
16 #define UART_MSR  6 /* In:  Modem Status Register */
17 #define UART_SCR  7 /* I/O: Scratch Register */
18
19 /*
20  * These are the definitions for the FIFO Control Register
21  * (16650 only)
22  */
23 #define UART_FCR_ENABLE_FIFO  0x01 /* Enable the FIFO */
24 #define UART_FCR_CLEAR_RCVR 0x02 /* Clear the RCVR FIFO */
25 #define UART_FCR_CLEAR_XMIT 0x04 /* Clear the XMIT FIFO */
26 #define UART_FCR_DMA_SELECT 0x08 /* For DMA applications */
27 #define UART_FCR_TRIGGER_MASK 0xC0 /* Mask for the FIFO trigger range */
28 #define UART_FCR_TRIGGER_1  0x00 /* Mask for trigger set at 1 */
29 #define UART_FCR_TRIGGER_4  0x40 /* Mask for trigger set at 4 */
30 #define UART_FCR_TRIGGER_8  0x80 /* Mask for trigger set at 8 */
```

```
31  #define UART_FCR_TRIGGER_14 0xC0 /* Mask for trigger set at 14 */
32  /* 16650 redefinitions */
33  #define UART_FCR6_R_TRIGGER_8 0x00 /* Mask for receive trigger set at 1 */
34  #define UART_FCR6_R_TRIGGER_16  0x40 /* Mask for receive trigger set at 4 */
35  #define UART_FCR6_R_TRIGGER_24  0x80 /* Mask for receive trigger set at 8 */
36  #define UART_FCR6_R_TRIGGER_28  0xC0 /* Mask for receive trigger set at 14 */
37  #define UART_FCR6_T_TRIGGER_16  0x00 /* Mask for transmit trigger set at 16
        */
38  #define UART_FCR6_T_TRIGGER_8 0x10 /* Mask for transmit trigger set at 8 */
39  #define UART_FCR6_T_TRIGGER_24  0x20 /* Mask for transmit trigger set at 24
        */
40  #define UART_FCR6_T_TRIGGER_30  0x30 /* Mask for transmit trigger set at 30
        */
41
42  /*
43   * These are the definitions for the Line Control Register
44   *
45   * Note: if the word length is 5 bits (UART_LCR_WLEN5), then setting
46   * UART_LCR_STOP will select 1.5 stop bits, not 2 stop bits.
47   */
48  #define UART_LCR_DLAB 0x80  /* Divisor latch access bit */
49  #define UART_LCR_SBC  0x40  /* Set break control */
50  #define UART_LCR_SPAR 0x20  /* Stick parity (?) */
51  #define UART_LCR_EPAR 0x10  /* Even parity select */
52  #define UART_LCR_PARITY 0x08  /* Parity Enable */
53  #define UART_LCR_STOP 0x04  /* Stop bits: 0=1 stop bit, 1= 2 stop bits */
54  #define UART_LCR_WLEN5  0x00  /* Wordlength: 5 bits */
55  #define UART_LCR_WLEN6  0x01  /* Wordlength: 6 bits */
56  #define UART_LCR_WLEN7  0x02  /* Wordlength: 7 bits */
57  #define UART_LCR_WLEN8  0x03  /* Wordlength: 8 bits */
58
59  /*
60   * These are the definitions for the Line Status Register
61   */
62  #define UART_LSR_TEMT 0x40  /* Transmitter empty */
63  #define UART_LSR_THRE 0x20  /* Transmit-hold-register empty */
64  #define UART_LSR_BI 0x10  /* Break interrupt indicator */
65  #define UART_LSR_FE 0x08  /* Frame error indicator */
66  #define UART_LSR_PE 0x04  /* Parity error indicator */
67  #define UART_LSR_OE 0x02  /* Overrun error indicator */
68  #define UART_LSR_DR 0x01  /* Receiver data ready */
69
70  /*
71   * These are the definitions for the Interrupt Identification Register
72   */
73  #define UART_IIR_NO_INT 0x01  /* No interrupts pending */
74  #define UART_IIR_ID 0x06  /* Mask for the interrupt ID */
75
76  #define UART_IIR_MSI  0x00  /* Modem status interrupt */
77  #define UART_IIR_THRI 0x02  /* Transmitter holding register empty */
78  #define UART_IIR_TOI  0x0c  /* Receive time out interrupt */
79  #define UART_IIR_RDI  0x04  /* Receiver data interrupt */
80  #define UART_IIR_RLSI 0x06  /* Receiver line status interrupt */
81
82  /*
83   * These are the definitions for the Interrupt Enable Register
84   */
85  #define UART_IER_MSI  0x08  /* Enable Modem status interrupt */
86  #define UART_IER_RLSI 0x04  /* Enable receiver line status interrupt */
87  #define UART_IER_THRI 0x02  /* Enable Transmitter holding register int. */
88  #define UART_IER_RDI  0x01  /* Enable receiver data interrupt */
89
90  /*
91   * These are the definitions for the Modem Control Register
92   */
```

```
93  #define UART_MCR_LOOP 0x10  /* Enable loopback test mode */
94  #define UART_MCR_OUT2 0x08  /* Out2 complement */
95  #define UART_MCR_OUT1 0x04  /* Out1 complement */
96  #define UART_MCR_RTS  0x02  /* RTS complement */
97  #define UART_MCR_DTR  0x01  /* DTR complement */
98
99  /*
100  * These are the definitions for the Modem Status Register
101  */
102 #define UART_MSR_DCD  0x80  /* Data Carrier Detect */
103 #define UART_MSR_RI 0x40  /* Ring Indicator */
104 #define UART_MSR_DSR  0x20  /* Data Set Ready */
105 #define UART_MSR_CTS  0x10  /* Clear to Send */
106 #define UART_MSR_DDCD 0x08  /* Delta DCD */
107 #define UART_MSR_TERI 0x04  /* Trailing edge ring indicator */
108 #define UART_MSR_DDSR 0x02  /* Delta DSR */
109 #define UART_MSR_DCTS 0x01  /* Delta CTS */
110 #define UART_MSR_ANY_DELTA 0x0F /* Any of the delta bits! */
111
112 /*
113  * These are the definitions for the Extended Features Register
114  * (StarTech 16C660 only, when DLAB=1)
115  */
116 #define UART_EFR_CTS  0x80  /* CTS flow control */
117 #define UART_EFR_RTS  0x40  /* RTS flow control */
118 #define UART_EFR_SCD  0x20  /* Special character detect */
119 #define UART_EFR_ENI  0x10  /* Enhanced Interrupt */
120
121 #endif /* _UART_H_ */
```

A.6.4 NoC.h

```
1  // Network/System specific values
2
3  #ifndef NOC_H
4  #define NOC_H
5
6  #define POOL_SIZE 128
7  #define POOL_SIZE_BIT 4
8  #define POOL_INVALID_INDEX -1
9  #define CTRL_RAY_TREES 10 // Should never exceed POOL_SIZE / NUM_LIGHTS
10
11 #define COMMUNICATION_SIZE 10 // Min 2
12 #define CTRL_IS_SIZE 30 // Min CTRL_RAY_TREES * NUM_LIGHTS
13 #define CTRL_SRG_SIZE 10 // Min CTRL_RAY_TREES
14 #define CTRL_SHADE_SIZE 10 // No min
15
16 #define sem_t int*
17
18 #define SEMAPHORE_BASE 0x40000000 // Base address of semaphores
19 // Semaphore 0 reserved for reset-code
20 #define SEM_PRG_COMM 14
21 #define SEM_IS_RAY_COMM 1
22 #define SEM_IS_HIT_COMM 2
23 #define SEM_SRG_GEN_COMM 3
24 #define SEM_SRG_RAY_COMM 4
25 #define SEM_SHADE_COMM 5
26 #define SEM_PRG_WAIT 6
27 #define SEM_IS_RAY_WAIT 7
28 #define SEM_IS_HIT_WAIT 8
29 #define SEM_SRG_GEN_WAIT 9
30 #define SEM_SRG_RAY_WAIT 10
31 #define SEM_SHADE_WAIT 11
```

```
32 #define SEM_JOB_ALLOCATE 12
33 #define SEM_MEM_ALLOCATE 13
34 #define SEMAPHORE_ADDRESS(i) (int*) (SEMAPHORE_BASE + 4 * i)
35 #define PASS_SEMAPHORE(x) while(!(*x)) {}
36 #define RELEASE_SEMAPHORE(x) (*x = 1)
37
38 #define FRAME_BUFFER_BASE 0xC0000000
39 #define RESOLUTION_X 32
40 #define RESOLUTION_Y 24
41 #define FRAME_BUFFER_ADDRESS(x, y) ((int*) (FRAME_BUFFER_BASE + x +
       RESOLUTION_Y * y))
42
43 #endif
```

A.6.5 ram.ld

```
1  MEMORY
2          {
3          vectors : ORIGIN = 0x00000000, LENGTH = 0x00001000
4          ram     : ORIGIN = 0x00001000, LENGTH = 0x0000f000
5          }
6
7  SECTIONS
8  {
9          .vectors :
10         {
11         *(.vectors)
12         } > vectors
13
14         .text :
15         {
16         *(.text)
17         } > ram
18
19         .data :
20         {
21         *(.data)
22         } > ram
23
24         .rodata :
25         {
26         *(.rodata)
27         } > ram
28
29         .bss :
30         {
31         *(.bss)
32         } > ram
33
34         .stack :
35         {
36         *(.stack)
37         _src_addr = .;
38         } > ram
39 }
```

A.6.6 reset.S

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

```
3
4     .global ___main
5     .global _offset
6           .section .stack, "aw", @nobits
7   .space  STACK_SIZE
8   .data
9     .align 4
10    .type _offset,@object
11    .size _offset,4
12  _offset:
13    .long 0
14
15  _stack:
16
17          .section .vectors, "ax"
18          .org 0x100
19
20    l.movhi r10,0x4000
21    l.ori r10,r10,0x0000
22
23  .L1:
24    l.lwz r11,0(r10)
25    l.sfeqi r11,0x0001
26    l.bnf .L1
27
28  _reset:
29    l.movhi r12,hi(_offset)
30    l.ori r12,r12,lo(_offset)
31    l.lwz r13,0(r12)
32          l.movhi r1,hi(_stack-4)
33          l.ori   r1,r1,lo(_stack-4)
34          l.addi  r2,r0,-3
35          l.and   r1,r1,r2
36    l.addi  r13,r13,0x2000
37    l.add r1,r1,r13
38
39    l.sw  0(r12),r13
40
41    l.sw  0(r10),r11
42
43    l.addi  r10,r0,0
44    l.addi  r11,r0,0
45    l.addi  r12,r0,0
46    l.addi  r13,r0,0
47
48          l.movhi r2,hi(_main)
49          l.ori   r2,r2,lo(_main)
50          l.jr    r2
51          l.addi  r2,r0,0
52
53  ___main:
54    l.jr  r9
55    l.nop
```