Mesochronous TDM-based Network-on-Chip

Anders la Cour Bentzon

Kongens Lyngby 2012
IMM-BSc-2012-13

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 IMM-BSc-2012-13

This thesis is typeset using LaTeX.

Abstract

Since wire delay makes it difficult to distribute a synchronous clock signal evenly in large digital systems, alternatives to the synchronous design paradigm are called for. This thesis proposes and implements a mesochronous router for a TDM-based network-on-chip. First, a synchronous router is designed, and a bi-synchronous FIFO is then introduced and its use as a synchroniser investigated. These FIFOs are used as synchronisers between the clock domains to make the router mesochronous. Finally, the design is verified to be working in practise as a proof-of-concept on an FPGA.

The solutions mentioned are analysed with regard to area, power consumption and speed, and clock-gated versions of the designs are proposed to reduce power. It is shown that while the mesochronous router works, it is in terms of area almost twice as large as a similar asynchronous router. Thus, the overhead incurred in a mesochronous system seems to favour an asynchronous approach.

Resumé (Danish)

Da forsinkelse i ledninger gør det svært at distribuere et synkront kloksignal jævnt i større digitale systemer, er det nødvendigt at finde alternativer til det synkrone designparadigme. Denne opgave implementerer en mesokron router for et TDM-baseret intrachip netværk. Først bliver en synkron router designet, og anvendelsen af en bi-synkron FIFO som synkroniseringsenhed undersøges. Disse FIFO'er bruges derefter som synkroniseringsenheder mellem klokdomænerne for at gøre routeren mesokron. Endelig bliver det efterprøvet, at designet virker i praksis ved at lave en implementation på en FPGA.

De nævnte løsninger analyseres med hensyn til areal, effektforbrug og hastighed, og klok-gatede versioner foreslås for at spare effekt. Det vises, at mens den mesokrone router fungerer, så er den arealmæssigt næsten dobbelt så stor som en lignende asynkron router. De omkostninger, som et mesokront system medfører, lader altså til at gøre en asynkron tilgang mere hensigtsmæssig.

Preface

Designing embedded systems, and in particular systems-on-chip, is an exciting area of research, because it requires that which is the essence of engineering: Creating a working, usable product that satisfies — maybe even astonishes — the end user, while complying with the numerous demands inflicted by the platform, which may dictate limitations on available space and power while insisting that the product run at top speed. These trade-offs are an integral part of engineering, and they are nowhere more pronounced than in embedded systems design.

In recent years, the tendency to connect together, on a single chip, several, heterogeneous processor cores has sparked increasing interest in research into the area which has now become known as networks-on-chip. The work presented here provides results for a particular network-on-chip component, and it is hoped that it will be used to compare the feasibility of this design with alternative solutions.

I would like to thank my friends, colleagues and family, who have endured and even supported me during the work of writing this thesis. In particular, I would like to express my gratitude to my supervisor, Professor Jens Sparsø of DTU Informatics, without whose guidance, patience and excellent advise this thesis would have been sorely lacking.

Anders la Cour Bentzon
Kongens Lyngby
June 2012

Contents

Chapter 1

Introduction

Networks-on-chip (NoC) address an issue increasingly faced in hardware design, and particularly in consumer electronics: How to connect several heterogeneous *intellectual property* (IP) cores together on the same chip, in a so-called system-on-chip (SoC), while maintaining a reasonable bandwidth between them, in a way that scales with the number of cores [BM06, HG11]. This is solved by letting the NoC provide a layer of abstraction, where each core communicates directly with a network adaptor, which then routes the communication packages through the network to the correct destination. In the NoC considered in this thesis, nodes are connected in a two-dimensional grid, with each node consisting of an IP core, a network adaptor and a router. Thus, the total bandwidth increases when the grid size increases. Packages are routed using a technique known as *virtual circuits*, by which a pre-defined route is established through the router nodes when two cores need to communicate; and this is scheduled using *time-division multiplexing* (TDM), where time slots are assigned beforehand in order to avoid blocking, and avoid arbitration in the circuits (see e.g. [DT04, DYN03]). Thus, a certain performance can be ensured beforehand, known as *guaranteed service*, which allows real-time processing, a feature that is important in many consumer electronics devices, such as set-top boxes that decode high-resolution video. Because offering real-time guarantees is relatively expensive — a time slot that is reserved, but currently not needed by its owner, remains unused, even if other packages are queued to be routed — some networks in addition provide a *best effort* layer, in which non-time-critical packages can be routed whenever there is free bandwidth.

There are numerous examples of different NoCs, and the research is on-going. Aethereal [GH10] and MANGO [BS05], respectively, are examples of a synchronous and an asynchronous NoC with guaranteed service and best effort using TDM. Aelite [HG11] is a mesochronous, simpler version of Aethereal; and [SS11] proposes an asynchronous router for an Aethereal-like network. The goal of this thesis is to provide a mesochronous version of the NoC router proposed by [SS11] in order to be able to make a reasonable comparison between the asynchronous and the mesochronous design paradigms as they relate to NoC development. Thus, performance indicators such as area costs, power consumption and speed are of particular interest as they are significant guideposts when it comes to deciding which implementation is most feasible.

NoCs are, like SoCs, normally implemented on application-specific integrated circuits (ASICs), as this is the best way to ensure the performance required of consumer electronics. Unfortunately, the ASIC design flow is nontrivial and time consuming, as well as expensive, so it lies outside the scope of a bachelor thesis. In order to still be able to have a target platform and to create a proof of concept, it has therefore been decided

to instead use an FPGA. In particular, the Digilent Nexys2 board will be used, which features the Xilinx Spartan3E-1200 FPGA along with several interfaces useful for testing, among these a seven-segment LED display. The Spartan3E-1200 has 1200K system gates, the equivalent of 19,512 logic cells, along with eight digital clock managers and 136K distributed RAM bits [Xil11]. This platform will be used when synthesising the implementations throughput the thesis, and the number of look-up tables (LUTs) required by a given design will be used as an estimate of die area. Finally, in Chapter 6, a single router will be synthesised, placed, routed and configured on this FPGA. To simulate the systems designed, ModelSim by Mentor Graphics will be used.

The theory presented in this thesis is not in itself overly complicated, and it has been attempted to introduce new concepts such that most readers familiar with electrical engineering at an undergraduate level should be able to follow along without resorting to other sources. However, there is a fine line between introducing and summarising a new concept and competing with textbooks to give the most thorough and theoretically satisfying explanation; the latter has deliberately not been attempted, so the reader may in some cases wish to refer to the relevant literature for a more in-depth treatment. As a starting point, [DP98] is an excellent textbook concerning digital systems, and most of the theory required in this thesis can be found in this book.

This thesis is divided into seven chapters. The chapter after this one provides a brief summary of the theory and background needed in the rest of the thesis. This is followed by a chapter describing the design and implementation of a simple Aethereal-like NoC router, which is a synchronous version of the one presented in [SS11]. Then a FIFO buffer is designed and its use as a synchroniser investigated, after which this is used to make a mesochronous NoC router. A simple test bench using this router is then implemented on an FPGA as a proof-of-concept, and finally the results obtained during the thesis are discussed, and areas of interest that need further work are proposed.

Chapter 2

Theory

This chapter provides a brief introduction to the theory and background required for the following chapters. The matters covered here are not intended to be exhaustive; rather, they should serve as useful summary, and the reader is advised to refer to the relevant literature for a more in-depth coverage.

First, an introduction to synchronisation issues and ways to synchronise between different clock domains is given, after which follows a brief overview of clock-gating methodology. Finally, a description of networks-on-chip and related concepts will be provided, along with an introduction to the network on which the rest of the thesis is based.

2.1 Synchronisation

Traditionally, the elements of a digital circuit are *synchronous* to the same clock signal, and the minimum clock period can be calculated as the worst-case time it takes a signal to propagate through the circuit and keep the minimum required flip-flop setup times. For the logic to work correctly, it is important that the clock signal is evenly distributed so that the clock 'ticks' at the same time in all the circuit elements. However, for large circuits, the efforts required to guarantee an even clock distribution increase prohibitively. A way to mitigate this is to divide the circuit into distinct *clock domains*, where each clock domain is locally synchronous, but where no effort is made to ensure that the clock domains are synchronous with each other. Since the clock signals originate from the same clock, the periods and frequencies are shared, but they thus have a (constant) phase difference; such circuits are termed *mesochronous*. However, in many practical situations, the wire propagation delay depends on a number of factors, significant among these temperature, so when the temperature changes unevenly across a mesochronous circuit (because of an uneven workload), the phase differences slowly drift. A system exhibiting this behaviour, with a slowly changing clock phase difference between its clock domains, is called *plesiochronous*. In the extreme end of the spectrum, the clock signal is completely removed, and circuit elements synchronise by other means, e.g. handshaking; such circuits are *asynchronous* [DP98, Chap. 10].

An important issue faced when working with non-synchronous circuits is how to *synchronise* between clock domains without incurring *metastability* [Gin11]. Metastability occurs when the input to a flip-flop changes *after* the setup time, which is to say when the input changes just before the clock ticks; when this happens, the flip-flop enters an indeterminate state and may eventually attain either the old or the new value, but after an arbitrarily long time, during which it is unusable. This is avoided in synchronous circuits,

because the clock period is determined with this in mind; but in non-synchronous systems, it is very important to synchronise signals traversing clock domains. A common way to do this is to use a bi-synchronous FIFO (*F*irst *I*n, *F*irst *O*ut) buffer, which is a memory element interfaced by two different clocks. Data is written to the FIFO synchronously to the write clock, and read from the FIFO synchronously to the read clock. A FIFO typically works by maintaining a data buffer that is synchronous to the write clock, and a write and read pointer synchronous to their respective clocks. The write pointer points to the element after the one just written, and the read pointer to the next one to be read; these pointers are incremented whenever data is written or read. In addition, the FIFO provides output signals to indicate whether the FIFO is full or empty, in which case data cannot be written or read, respectively. Figures characterising a FIFO are its width — the size of a data word — and its depth, which is number of words it can contain.

2.2 Clock-gating methodology

When considering the power consumption of an electrical circuit, a significant amount of this is caused by switching activity; when a signal goes from low to high, energy is required to charge the capacitive load of that signal. Thus, power consumption can be reduced by limiting unnecessary switching, but in clocked circuits, the regular activity of the clock causes energy to be dissipated in the clock inputs of registers (flip-flops), even when the actual contents of those do not change. A way to avoid this is to gate the clocks, that is, to disable clock signals for parts of the system when those parts are not in use — effectively turning those parts off. [Aro12, Section 2.5] describes different ways to do this, and in particular introduces the standard clock-gating cell of Figure 2.1. When the `enable` input is high, the clock signal (`clk`) is propagated to the gated clock output (`gatedClk`); but when `enable` is low, `gatedClk` remains low, no matter the value of `clk`. Since it is important to maintain a stable clock frequency, care has to be taken not to cut off the clock signal prematurely, which is the purpose of the latch; this makes it possible to change `enable` at any time while guaranteeing that `gatedClk` will always be high for precisely one half clock period at a time. Thus, if `enable` is disabled while `clk` is high, `gatedClk` remains high until `clk` goes low.

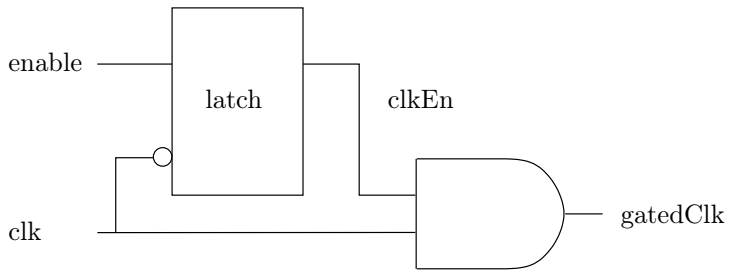

Figure 2.1: Standard clock-gating cell without test signal [Aro12, Fig. 2.26]

2.3 On-chip interconnect

Since this thesis deals only with the design of a mesochronous NoC router based on the asynchronous router presented in [SS11], it does not consider issues which lie beyond the router hardware, such as network adaptors, scheduling, configuration and so forth. Thus, only concepts pertinent to the immediate router design will be covered here.

Data arrives at a router in *packages*, where a package consists of a number of *flits* (flow-exchange digits). Each flit is a 35-bit word according to Table 2.1, consisting of 32 bits of data followed by bits signalling end of package (EOP), start of package (SOP) and valid data. The first flit in each package is a header flit, with a high SOP bit, where the data field contains routing information describing how this package is to be routed

to its destination. Subsequent flits in the package contain 32 bits of actual data, and the package is terminated by a flit whose EOP bit is high. Flits which are part of a package have a high valid bit; this is to easily distinguish them from signals between packages.

Table 2.1: Flit format

Bit	34	33	32	...	1	0
Description	valid	SOP	EOP	data	data	data

Packages are routed according to the address information of the header flit. A router decides which output port to route a package to based on the first two bits of the header flit, according to Table 2.2 (see Figure 3.1 for the physical layout of the router). Before the header flit is sent to the output port, its address field is shifted two bits right so that the new leading bits contain routing information for the next hop in the route. If the package is destined for the local IP core, the address bits are those of the port from which the package originates (thus, a package arriving from the North port, whose first two address bits are 00, are routed to the local port, and *not* back to the North port). A package in the router of [SS11] consists of three flits, which is adopted in the router presented here. However, during many of the simulations, when testing the functionality, only two flits will be routed per package in order to keep the wave window uncluttered.

Table 2.2: Address format

North	00
East	01
South	10
West	11

Chapter 3

The Synchronous Network

This chapter describes a reference implementation of a simple network-on-chip router. It is intended to be a synchronous version of the asynchronous router described in [SS11], which is based on the Aetherial network [GH10]. Thus, the design in this chapter will serve to gain a useful, initial understanding of the concept, and it will provide data which can be used as a reference when compared to the more advanced solutions presented later.

First, a simple implementation of the router will be described and analysed, and afterwards this router will be clock gated to minimise its power consumption when it is not in use.

3.1 Simple router

As described in the previous chapter, the basic building block of the network is the *router*. This section describes the design of such a router and its subcomponents; then the router is synthesised and simulated to verify its functionality, and its power consumption is analysed.

The network is conceptually organised in a two-dimensional grid, so that each router has four neighbours. Furthermore, each router is connected to a local IP core, which contributes to a fifth port. In this design, these ports are referenced as shown in Table 3.1; please also refer to Figure 3.1.

Table 3.1: Convention for physical port numbers

0	South
1	West
2	North
3	East
4	Local

3.1.1 Router

The conceptual design of a router is shown in Figure 3.2.[1] The router consists of five input and five output lines which are connected with a crossbar. A header parsing unit (HPU) parses the information in each line and generates control signals for the crossbar that

[1] Please refer to the file `router.vhd` in Appendix A.1 for the VHDL implementation of the router.

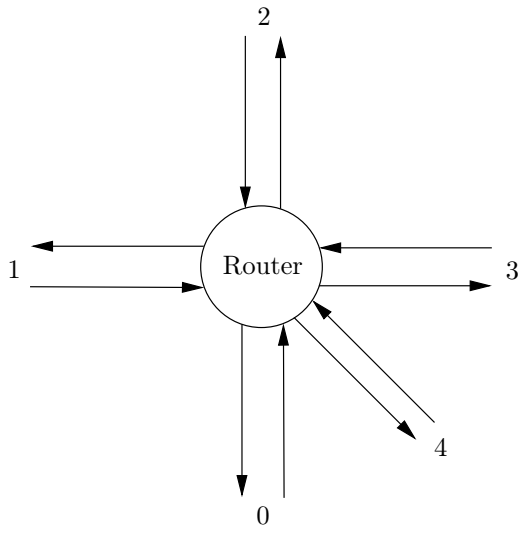

Figure 3.1: Convention for physical port numbers

ensure that each flit is delivered to the correct output line. To increase throughput, it is pipelined in two stages as shown in the figure. This pipeline depth was chosen because the synchronisers, which will be added in Chapter 5, have a latency of one clock cycle; thus, it is effectively a three-stage pipeline, which corresponds well with the chosen package size of three flits (and conversely, [SS11] uses three-flit packages because a pipeline depth of three is appropriate).

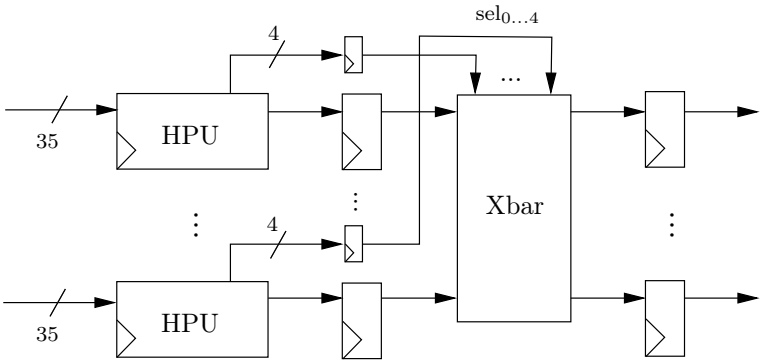

Figure 3.2: Generic block diagram of the synchronous router

3.1.2 Crossbar

As in [SS11], the crossbar is controlled with a one-hot encoded signal as depicted in Table 3.2.[2] For example, to route the signal on input port 4 to output port 1, the MSB should be set. The crossbar is designed to route the incoming signal to the output port as determined by the select signal, and to output logical 0's on any port not connected to an input port.

The one-hot encoding makes it possible to demultiplex input signals using simple *and* gates. The output ports are then multiplexed using *or* gates, which ensures that the entire crossbar consists of only two layers of gates (see Figure 3.3). This is very simple to design and should ensure a reasonably low propagation delay. It also means that, since the control signal is ordered by the source port, the full control signal can be generated

[2] Please refer to the file `xbar.vhd` in Appendix A.1 for the VHDL implementation of the crossbar.

simply by concatenating the contributions of each HPU. Note that the output is undefined if two input signals are routed simultaneously to the same output port.

Table 3.2: The 20-bit one-hot control signal for the crossbar

Source port	4	3	2	1	0
Destination port	1032	1042	1034	4032	1432
	MSB				LSB

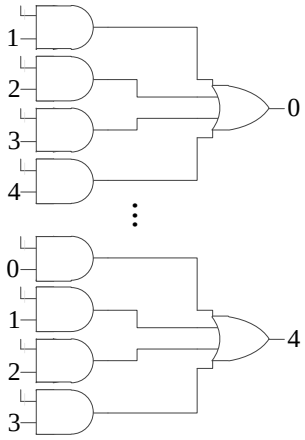

Figure 3.3: Diagram of the crossbar

3.1.3 Header parsing unit

The header parsing unit is depicted in Figure 3.4.[3] Its purpose is to decode the address information of the first flit in each package and generate an according control signal to the crossbar, so that all three flits of the package are routed to the correct destination; this is done using a simple binary decoder. Thus, the two-bit address field is decoded into a four-bit one-hot signal as shown in Table 3.2. Also, as described in Section 2.3, the address information in the first flit is shifted two bits. When `SOP` is high, the decoded crossbar select signal is saved in the register, and it remains there until `EOP` is high, at which point the register is reset with 0's.

3.1.4 Synthesis

Synthesising this router for a Xilinx Spartan3E FPGA reveals that it requires a total of 390 flip-flop bits; please see Table 3.3.[4] Furthermore, the synthesis report shows that the router requires 414 slices (4%) and 761 four-input LUTs (4%).

It is a bit unexpected that the router requires significantly more LUTs than flip-flops, so to investigate this further, the HPU and crossbar are synthesised separately.[5] Each of the five HPUs requires 48 LUTs and four flip-flops, while the crossbar alone requires 525 LUTs. This adds up to 765 LUTs, which is actually four more than the router as a whole. The router itself contains no real logic, and it is feasible that the synthesiser has been able to optimise a bit when connecting the components together. The conclusion seems to be that the main consumer of LUTs is the crossbar, which is completely combinational.

[3] Please refer to the file `hpu.vhd` in Appendix A.1 for the VHDL implementation of the HPU.
[4] Please refer to the file `router.syr` in Appendix B.1 for the Xilinx XST synthesis report.
[5] Please refer to the files `HPU.syr` and `Xbar.syr` in Appendix B.1 for the synthesis reports.

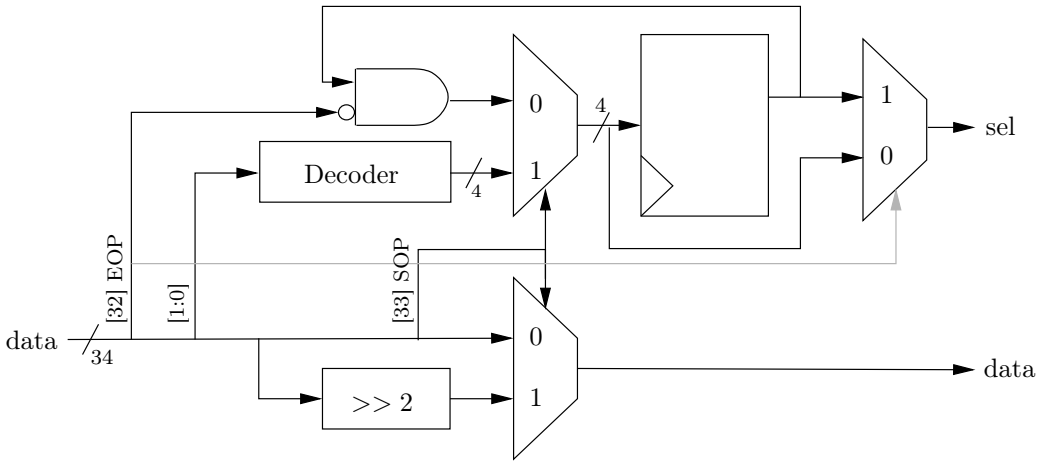

Figure 3.4: Diagram of the header parsing unit

Table 3.3: Register count for the synchronous router

Description	Count	Bits
35-bit pipeline register (data)	10	350
20-bit pipeline register (select signal)	1	20
4-bit address register (HPU)	5	20
	16	390

The timing report (which is only an estimate, since the design was not placed and routed) shows that the critical path is through the crossbar, with a minimum period of 3.9 ns corresponding to a maximum frequency of 257 MHz. That the critical path lies here confirms the value of using a simple crossbar without too much complexity; and this is indeed a reasonable speed. It could probably be only marginally increased by introducing a pipeline register through the middle of the crossbar between the layers of *and* and *or* gates.

3.1.5 Simulation

A test environment is generated by supplying each router input port with a new flit according to a predefined test vector stipulating which packages are to be sent at which state in the test.[6] A 'package' consists of a header flit containing the destination address, and a stop flit containing a sequence number. The test vector is defined so that all output ports are tested, and the test is run so that the same test vector package is sent through all the input ports in turn.

Similarly, in another process, the output of the router is read, and the data is compared to the test vector. A warning is generated if an unexpected flit arrives, if no flit arrives when one is expected, or if the sequence number doesn't match.

In the simulation in Figure 3.5, a package (consisting of a header flit and a data/stop flit) is sent to port 0 (bottom of the picture) from all input ports (middle of the picture). As can be seen, all the packages arrive at output port 0, except for the one sent from input port 0, which arrives at the local output port (4), as expected. Also, there is a latency of two clock periods, due to the router's pipeline depth of two. Note that the address information of the first flit of each package is removed; actually, the entire address field is right-shifted by two bits in accordance with the design of the HPU (see Figure 3.4). Also note that the sequence numbers of the received data flits match those of the submitted flits.

[6] Please refer to the file `testRouter.vhd` in Appendix A.1 for the VHDL implementation of the test bench.

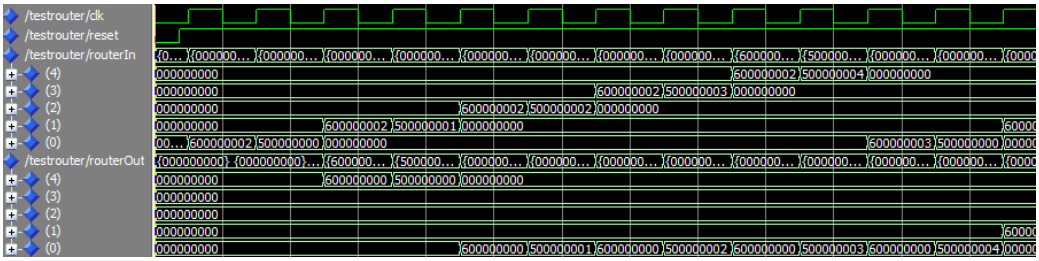

Figure 3.5: Simulation of the synchronous router

3.1.6 Power consumption

Measuring power consumption for the router presented above is not trivial. For one thing, it depends significantly on the usage scenario; and for another, it requires advanced simulation tools and techniques. [AJI07] describes a way to measure power consumption for systems on an FPGA, but even though the target platform in this thesis is indeed an FPGA, the system is intended to run on an ASIC, so this is not really interesting. To estimate power usage for an ASIC, a tool such as Synopsis would have to be used, which is unfortunately outside the scope of this bachelor thesis.

Nonetheless, a very rough estimate is still useful in order to compare the different router designs presented in this thesis. As such, it is the relative power consumption of the different designs that is of interest. Thus, focus will rest on the switching power that is consumed when driving the signals from low to high. ModelSim can record a toggle count, which is a representation of switching power, for most signals; however, ModelSim does not record toggles of the clock signals that drive the flip-flops, even when the flip-flop contents do not change. Unfortunately, it also turned out that ModelSim only records whether or not a given signal has toggled, and not how many times this has happened, making this number useless as an estimate of switching power.

Since later parts of this thesis focus on minimising the power consumption of inactive flip-flops, the main measurement of interest is the power reduction due to this adjustment, and an estimate of this can be obtained by manually counting the number of active flip-flops. Since this is only a rough estimate, no further analysis of the different capacitive loads or the fan-outs will be taken into account, and this figure will simply be interpreted as a relative benchmark of the total power consumption. As the main goal of this benchmark is to compare different designs, its accurateness is of minor importance as long as the same procedure is used to generate it for each design and it does not significantly bias one of the designs.

At the same time, this analysis needs to be carried out on a realistic and typical usage scenario. This is close to impossible without knowing more of the exact application of the network-on-chip, so it is chosen somewhat arbitrarily to presume that a given router will be in use about 20% of the time. The package size will be three flits to correspond with [SS11]. Table 3.4 shows a usage scenario in which three packages (totalling nine flits) are routed through the router during ten time slots. This consumes nine routes out of a total of 50, so this router can be said to be in use 18% of the time, which corresponds well enough with the 20% mentioned above. Notice that some of the time, the router is only used to process a single flit; and during some time slots, it is not used at all. Thus, this usage scenario favours a router that is able to reduce its power consumption when it is almost inactive, and when it is completely inactive; this seems realistic enough. It should be mentioned that the pipeline depth of the router means that it takes more than one time slot for a package to finish processing; Table 3.4 refers to the input ports of the router[7]

Referring to Table 3.3, the router consists of 390 flip-flops whose clock signals toggle from low to high once for each time slot (clock cycle), so its power consumption totals

[7]Please refer to the file `testPower.vhd` in Appendix A.1 for the VHDL implementation of the power consumption test bench.

Table 3.4: Power estimation of the simple router

Time slot	1	2	3	4	5	6	7	8	9	10
1st package	0–3 (start)	0–3 (data)	0–3 (end)							
2nd package		1–4 (start)	1–4 (data)	1–4 (end)						
3rd package	1–0 (start)	1–0 (data)	1–0 (end)							
Flip-flops	390	390	390	390	390	390	390	390	390	390

3900 clock toggles as shown in Table 3.4.

3.2 Clock-gated router

Because of the pipeline registers, the router presented above uses power even when it is not in use. Switching power loss occurs whenever a signal is driven high, so by forcing the signals to be constantly zero when not used, some of this is avoided (another strategy could be to let them keep their last value). However, the clock signals drive the capacitive load of the pipeline register flip-flops even when they contain no useful data. A way to mitigate this is to turn off the clock signal when nothing is routed; a system using this approach will be presented in the following section.

Clock gating is a technique used on ASICs to minimise power consumption, but since FPGAs use special-purpose wiring for the clock signals to minimise skew, it is not recommended to use standard clock-gating approaches on FPGAs. Instead, one may use special vendor primitives, such as the Xilinx Digital Clock Managers or the like, which are technology dependent. Even though the Spartan3E FPGA is used as the target platform in this thesis, clock gating will be investigated in order to analyse the hardware from a more generic perspective, and synthesis results (mainly LUT count) will be presented as an estimate of area utilisation. The clock-gated circuits should not, however, be implemented on FPGAs.

3.2.1 Clock-gating strategy

While the NoC router presented in the previous section does not make any presumptions as to the nature of the data that is routed, and the way this happens, a typical usage scenario will probably dictate that a particular link is only used about 20% of the time. It is therefore highly desirable to design the system in such a way that it limits the amount of power consumed when it is not used.

With this in mind, the simplest approach is to monitor all the signals at the input ports of a given router and turn off its clock signal if none of them is valid. In order to determine whether the input signal at a given port is valid, a 35th bit is introduced in the flit format; this bit is high whenever the flit contains a valid data signal (see Table 2.1). A similar flag could be generated using a simple state machine by exploiting the start of package and end of package bits.

Figure 3.6 depicts a clock-gated synchronous router. On the basis of the incoming data signal, a clock-gating circuit determines whether or not the clock should be kept on. The clock signal generated by this circuit is distributed to the components of the router.

In the above approach, it can be determined when the data produced at the input ports is no longer valid; but this does not indicate whether the consumer has read all the data. Since the latency through the router is two clock periods, the clock-gating circuit can simply wait two clock cycles after detecting an invalid input signal before gating the clock. Using the standard clock-gating cell in Figure 2.1 ensures that the clock is not turned off prematurely, guaranteeing that a full clock signal is generated. Figure 3.7

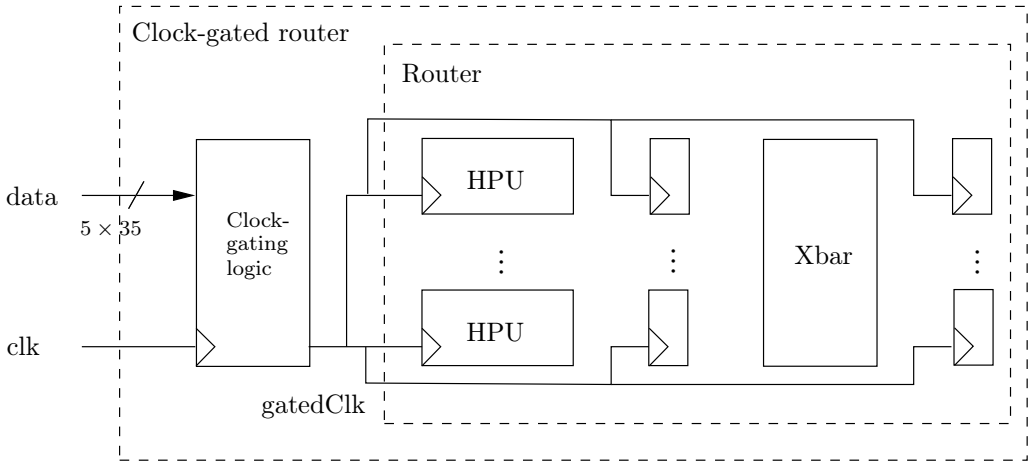

Figure 3.6: Clock distribution for the clock-gated router

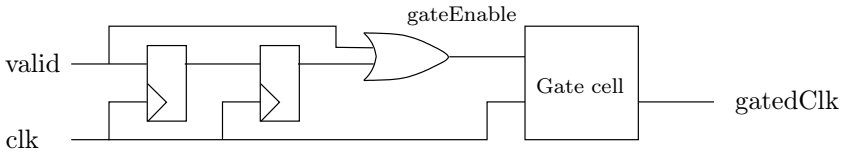

Figure 3.7: Clock-gating logic, two-period latency

illustrates the circuit used to gate the clock (this may fail if only a single valid data flit arrives; however, we presume that they arrive in packages of three).

While it may be tempting to further fine-tune the clock gating by turning off individual lines in the router when these are not used, this is not as easy. For one thing, it would interfere with the pipeline, and care would have to be taken to ensure consistency of the data; and for another, the data is interwoven after the crossbar, so the enable signal would have to depend on the crossbar select signal generated by the HPUs. When considering that the clock-gating logic, while cheap, is not completely free, and that the flip-flops used here consume power all the time, it is deemed that a more fine-tuned approach is probably not worth the effort; but of course, this depends the exact use scenario of the router. Also, it should be noted that the logic used to generate the clock enable signal cannot take more than half a clock cycle to do this, otherwise the clock won't be turned on in time [Aro12, p. 31]; this puts an additional contraint on how complicated it can be.[8]

3.2.2 Synthesis

The synthesiser reports that the clock-gated router uses 416 slices (4%) and 764 four-input LUTs (4%), which is only slightly more than the simple router (414 slices and 764 LUTs).[9] In addition to the registers used by the router itself, the clock-gating logic needs two flip-flops for implementing the delay as depicted in Figure 3.7 and one latch as per Figure 2.1, for a total of 392 flip-flops and 1 latch (see Table 3.5. The maximum frequency is 256 MHz (3.9 ns), which is virtually the same as before; the critical path is still through the crossbar. Furthermore, it is reported that the clock-gating circuit itself has a minimum period of 2.1 ns corresponding to a maximum frequency of 476 MHz. This means that the actual maximum frequency at which this circuit should be clocked is 238 MHz.

[8] Please refer to the file `gatedRouter.vhd` in Appendix A.1 for the VHDL implementation of the clock-gated router.

[9] Please refer to the file `gatedRouter.syr` in Appendix B.1 for the Xilinx XST synthesis report.

Table 3.5: Register count for the clock-gated synchronous router

Description	Count	Bits
35-bit pipeline register (data)	10	350
20-bit pipeline register (select signal)	1	20
4-bit address register (HPU)	5	20
Clock-gating register (2-period delay)	1	2
Latch (clock-gating cell)	1	1
	18	393

3.2.3 Simulation

The clock-gated router is tested using the same test bench in Section 3.1.5; the test vector is simply changed to provide an inactive period in the middle of the test where no data is routed. Figure 3.8 shows the router inputs and outputs, as well as the gated clock signal, when the input signal becomes invalid (that is, no package data is supplied). As can be seen, the clock-gating circuit allows enough time for the last flit to be processed through the pipeline from input port 4 to output port 1 before turning off the clock; the clock-gating logic of Figure 3.7 disables the `gateEnable` signal after two clock periods, and the standard clock-gating cell (Figure 2.1) turns off the `clkEn` latch on the falling clock flank, ensuring that the clock signal is not cut off.

Figure 3.8 also shows that the latency of two clock cycles in the clock-gating circuit means that the clock remains active for one period after the last valid signal has been processed, which effectively makes sure that the inactive signal is routed through to the output of the router. A more aggressive strategy would be to not allow this signal through, which would make it possible to turn the clock off one cycle earlier; but in this case, the latest valid signal would be kept at the output of the router, so the consumer would need to be able to detect that.

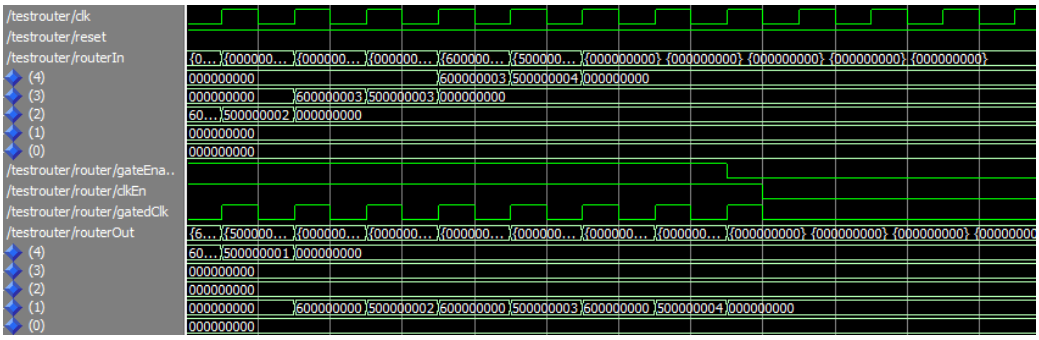

Figure 3.8: Simulation of clock-gated router when clock is turned off

Similarly, Figure 3.9 shows how the clock-gating circuit detects a new incoming signal and turns on the clock again. This happens in time for the router to process the first signal; as shown, the first flit is routed successfully from input port 0 to output port 2.

3.2.4 Power consumption

When analysing power consumption, the clock-gated router uses roughly the same amount of power as the simple router, except when it is completely inactive. It has a total of 393 flip-flops (and latches), of which 390 are clock gated. Figure 3.10 shows a simulation of the router when subjected to the usage scenario of Table 3.4, and in particular the gated clock signal (top of the figure). Referring to Figure 3.2, it can be seen that the first (HPU) pipeline register is not turned on until the end of the first pipeline stage, at which point the output of the HPU stage is clocked into this register. The router then remains active

Table 3.6: Power estimation of the clock-gated synchronous router

Time slot	1	2	3	4	5	6	7	8	9	10
1st package	0–3 (start)	0–3 (data)	0–3 (end)							
2nd package		1–4 (start)	1–4 (data)	1–4 (end)						
3rd package	1–0 (start)	1–0 (data)	1–0 (end)							
Flip-flops	3	393	393	393	393	393	393	3	3	3

until the eighth time slot. As shown in Table 3.6, it can be seen that the synchronous router thus has a total of 2370 flip-flop toggles.

3.3 Results

In this chapter, a simple synchronous router consisting of five header parsing units connected to a crossbar was designed and implemented. It was synthesised to estimate its area cost and timing parameters, and it was simulated in ModelSim to verify its functionality. Furthermore, a strategy was proposed to clock gate this router, and this was carried out and simulated as well. Power consumption was estimated for both designs on the basis of a usage scenario where the router is used 18% of the time and is measured by the amount of low-to-high clock ticks that drive flip-flops during a standard time interval of 10 time slots (clock cycles). Table 3.7 shows the results obtained in this chapter.

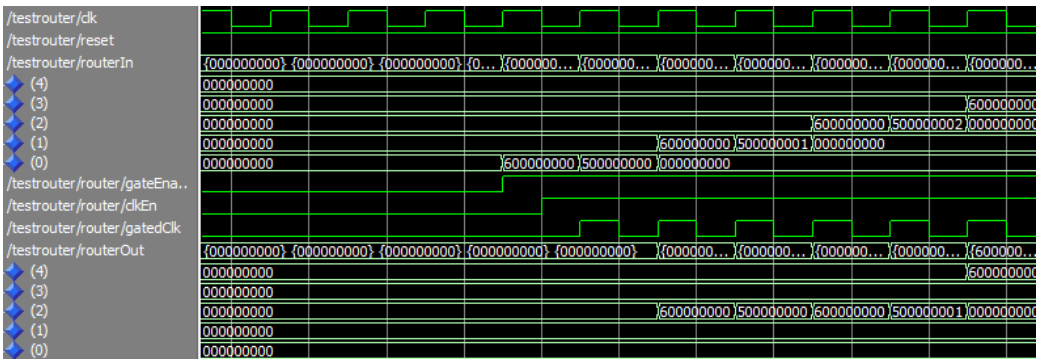

Figure 3.9: Simulation of clock-gated router when clock is turned on

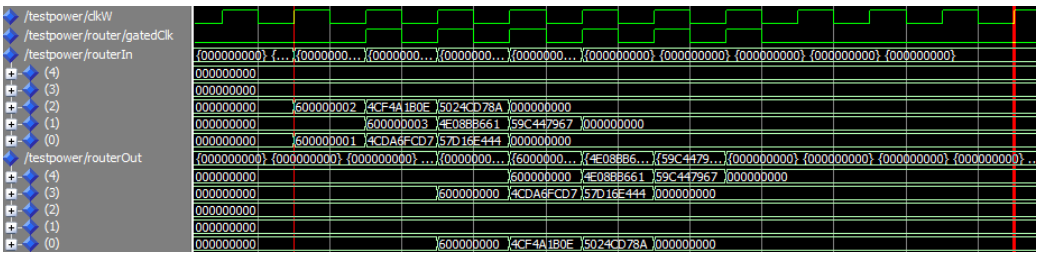

Figure 3.10: Analysis of power consumption for the synchronous router

Table 3.7: The results obtained for the synchronous router

Free running				Clock gated			
LUTs	Flip-flops	Power	Frequency	LUTs	Flip-flops	Power	Frequency
761	390	3900	257 MHz	764	392	2370	238 MHz

Chapter 4

A FIFO Synchroniser for Mesochronous Networks

In this chapter, a FIFO buffer is introduced in order to facilitate synchronisation between neighbouring nodes in a large mesochronous network. Originally, it was intended to use an 'off-the-shelf' solution and incorporate it into the proposed network without spending a great deal of effort trying to understand the intricate inner workings of the FIFO; but while working with this component, it turned out that using it is not as trivial as it first seemed, and its behaviour warranted a more thorough investigation. This chapter is dedicated to understanding the FIFO and the problems incurred in using it in a mesochronous system.

First, a third-party FIFO buffer design is described and analysed; then, an improvement to the full detector of this FIFO is proposed and implemented, and its results verified; and finally, the FIFO buffer is clock gated in order to minimise the power it consumes when it is inactive.

4.1 Bi-synchronous FIFO synchroniser

To synchronise between neighbouring routers, the bi-synchronous FIFO design described in [MPG07] will be used. This offers the benefits of having been already tested and incorporated in the DSPIN network-on-chip [MPGS06, MPCVG08], which means that it

- is designed to be interfaced by two synchronous systems with independent clock frequencies and phases;
- promises to be relatively inexpensive in terms of area; and
- is technology independent, so that it can be used on different FPGA architectures as well as on ASICs using standard cells.

Thus, it seems a reasonable choice for a synchroniser for the network presented in this thesis. The reason for using a FIFO as a synchroniser, and not just a couple of normal registers as described in [Gin11] is that the FIFO offers a better tolerance for clock skew; this will be investigated in Section 5.2.

4.1.1 Design

The main contribution of [MPG07] is to propose using a *token ring* to 'bubble-encode' the read and write pointers of the FIFO. This is done in order to ensure usability if metastability occurs when synchronising the token ring to another clock domain, as depicted in

Figure 4.1 (this figure is copied from [MPG07]). Thus, for a FIFO of depth N, the pointer is an N-bit word, and the position of the pointer is indicated by a two-bit token. For example, for $N = 5$, the token ring may be 00011. To increment this pointer, it is shifted (rotated) right by one position, so it becomes 10001; this ensures that one of the token bits remains constant during each operation, so it is guaranteed to be free of metastability when synchronised. Thus, the result of the synchronisation is never completely useless (if metastability were to occur, it could result in either 00001 or 10011, but never in 00000). By convention, the position of the write pointer is defined to be that of the second token bit, while the position of the read pointer is the one after the second token bit — see Table 4.1.[1]

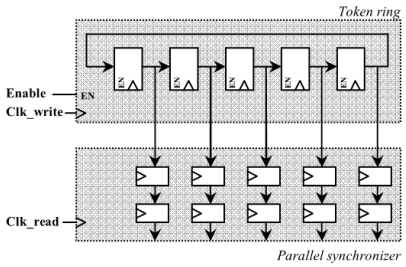

Figure 4.1: Synchronisation of a token ring [MPG07, Fig. 2]

Table 4.1: FIFO status and read/write pointers

Write pointer	0001**1**	**1**0001	**11**000	01**1**00	00**1**10	0001**1**
Read pointer	011**0**0	011**0**0	011**0**0	011**0**0	011**0**0	011**0**0
Number of elements	Empty	$N-4$	$N-3$	$N-2$	$N-1$	N

As can be seen in Table 4.1, the write pointer is incremented by one by shifting it right one bit each time an element is written to the FIFO, and likewise for the read pointer when an element is read. The pointers are initialised to the left-most situation, which thus indicates an empty FIFO. However, this is indistinguishable from its containing N elements as depicted in the right-most column. To solve this problem without having to maintain an extra status register — which adds complexity to the full and empty detectors — [MPG07] defines the FIFO to be full when it contains $N-1$ elements so that the N-element situation will never occur.

The empty detector in this FIFO is designed to raise a flag when the token rings are aligned as in the left-most column of Table 4.1. Since the empty detector resides in the domain of the read clock, it must synchronise the write pointer using a synchroniser as in Figure 4.1. It then operates by detecting a transition between a 0 and a 1 in the synchronised pointer (which is guaranteed to be present because of the bubble encoding), and asserts empty if this transition occurs in the position relative to the read pointer as shown in Table 4.1.

The full detector could work in a similar way, but in order to reduce area costs, [MPG07] proposes a simpler version. By *and*'ing the two pointers without synchronisation and collecting this in an *or* gate, it detects the $N-3$ and $N-2$ (defined as 'quasi-full') as well as the $N-1$ situations. This signal is then synchronised to the write pointer clock domain. Because of the synchronisation latency, this full detector needs to predict the full condition by also detecting the quasi-full situations. Since this sometimes prevents the FIFO from being completely filled, an improvement is proposed which allows writing to the FIFO for one extra cycle if the sender was not writing when the full signal was first asserted.

The FIFO is originally designed to interface two asynchronous clock domains, but [MPG07] proposes a mesochronous adaption, by which the FIFO is simplified by removing

[1] Please refer to [MPG07] for elaboration.

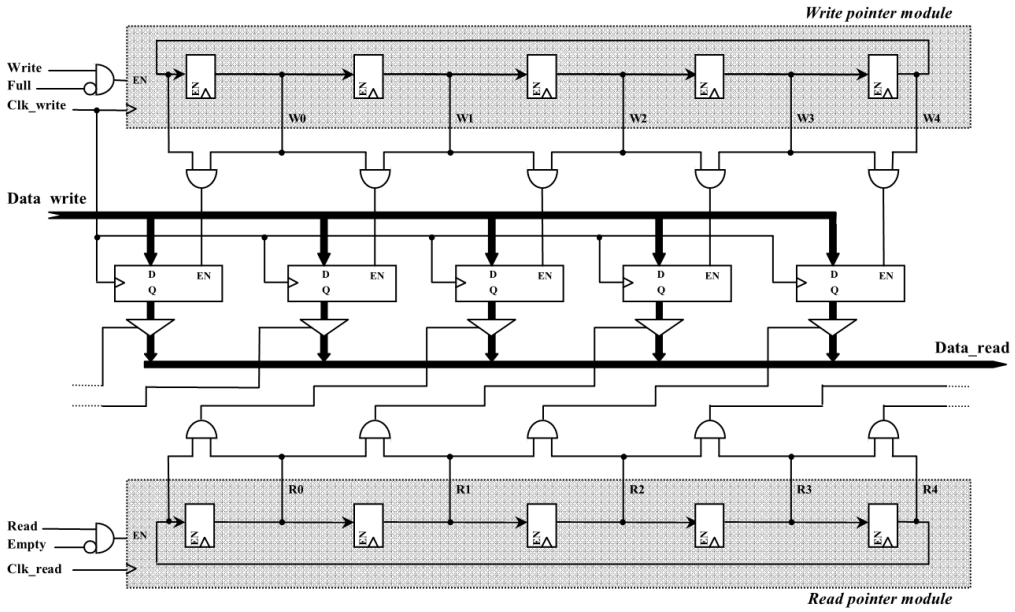

Figure 4.2: Diagram of the FIFO [MPG07, Fig. 7]

one of the synchronisation register rows of Figure 4.1; this reduces the latency as well as the area costs. Since the rising edge of the read clock is predictable in a mesochronous system, the bottom row of registers in Figure 4.1 is not needed if the top row is clocked so that no metastability occurs when synchronising the data; this can be achieved either by using a delayed version of the read clock, or by making the phase difference between the read and write clocks between 90° and 270° degrees. In the DSPIN network, this is accomplished by clocking neighbouring nodes with a 180° phase difference.

[MPG07] notes that a non-optimal full detector does not penalise throughput as much as a non-optimal empty detector, which is why the above simplification is reasonable; but a consequence is that, for FIFOs with a depth of less than six in an asynchronous system and five in a mesochronous system, throughput is only 50%. This will be confirmed in the simulation.

Figure 4.2, which is borrowed from [MPG07], shows the layout of the FIFO. The top is the write pointer, which as shown is synchronous to the write clock domain, and the bottom is the read pointer, which is synchronous to the read clock domain. In the middle, the data buffer, synchronous to the write clock domain, is shown. Using *and* gates, the two pointers are converted to a one-hot encoded signal, which is used to enable the correct register for writing, and to select from amongst a set of tri-state buffers the right register output for reading. When the write enable signal is applied, data is written to the next data buffer register, and the writer pointer is rotated, as long as the full signal is not high; and likewise, the read pointer is only rotated if the empty signal is not high.

4.1.2 Implementation

The FIFO was implemented in VHDL based on [MPG07].[2] Because it is intended to be part of a mesochronous, and not asynchronous, network, one of the synchronisation register rows in Figure 4.1 was removed as described in the article. The non-optimised full detector was improved with the adaption described above, so that the full detector delays raising its full flag for one clock cycle if the producer was not writing continuously at the time the full condition occurred.

[2] Please refer to the files `fifo.vhd`, `tokenring.vhd`, `fullDetector.vhd` and `emptyDetector.vhd` in Appendix A.2.

The data buffer was inferred as normal registers (flip-flops), and a multiplexer was used to select the output signal from amongst the data buffer registers instead of the tri-state buffers suggested in [MPG07], since the Spartan3E FPGA does not feature tri-state buffers.

To ensure 100% throughput, a FIFO depth of five was chosen, with a width of 35 bits to accomodate the flit size of the network.

4.1.3 Synthesis

The Xilinx synthesiser reports that a single FIFO requires 193 flip-flop bits, as shown in Table 4.2.[3] It uses 167 slices (1%) and 213 four-input LUTs (1%). The synthesiser finds the critical path to be through the full detector and calculates the minimum clock period as 5.30 ns, corresponding to a frequency of 189 MHz.

Table 4.2: Register count for the bi-synchronous FIFO

Description	Count	Bits
5-bit register (token rings)	2	10
5-bit synchronisation register (empty detector)	1	5
1-bit synchronisation register (full detector)	3	3
35-bit data buffers (FIFO)	5	175
	11	193

Synthesising the components individually reveals that each token ring requires only one LUT; the full detector requires six LUTs; and the empty detector eight LUTs.[4] Thus, the vast majority of the LUTs are spent implementing the multiplexer which is used to select the output data signal.

4.1.4 Simulation

To verify the functionality of the FIFO implementation, a test bench was created that would continuously write values to the FIFO and simultaneously read them again.[5] The read and write operations were simulated to originate from two different, phase-opposite clock domains.

Figure 4.3 shows the result of simulating a FIFO of depth four. Data is continuously written to the FIFO as long as it's not full, and continuously read as long as it's not empty. As can be seen, the correct data is retrieved in the correct order. However, it is immediately obvious that, as predicted, the throughput is only 50%. A closer look reveals that it is caused by the latency in the full detector: After the third element has been written, writing stops because the FIFO is reported as full. However, at this point, the first value has already been retrieved, and the second is on the way. All the same, the full detector asserts the full signal for three clock periods, at which point the FIFO has been completely emptied. Thus, the entire process is stalled. This happens repeatedly every three writes. It should be noted that the empty detector always gives the correct signal.

When simulating a FIFO of depth five, as shown in Figure 4.4, this does not occur. The extra element ensures that the full flag is not raised after the third write, as in Figure 4.3. But why not after the fourth? What happens 'behind the scenes' is that, in Figure 4.3, the FIFO is actually detected as full after the first write (when it contains $N - 3 = 1$ element), but because the full detector has a latency of two clock periods, this is not asserted until after the third write. Similarly, in Figure 4.4, the FIFO is internally detected as full just after the second write (when it contains $N - 3 = 2$ elements), but this only lasts for half a clock cycle; then the change in the read pointer is detected, and the full detector deasserts the internal full flag. In the first instance, there's simply not enough time for this change in the read pointer to be picked up.

[3] Please refer to the file `fifo.syr` in Appendix B.2 for the Xilinx XST synthesis report.
[4] Please refer to the files `tokenring.syr`, `fullDetector.syr` and `emptyDetector.syr` in Appendix B.2.
[5] For the VHDL implementation of the test bench, see the file `testFifo.vhd` in Appendix A.2.

The simulation also illustrates that while writing happens synchronously on the rising clock edge, the read functionality is combinational and transparent; as soon as the read enable signal is asserted, the data appears on the output (after a propagation delay, of course). Only when the read enable signal is asserted on the rising clock edge of the read clock is the read pointer incremented, however.

These tests thus confirm that, due to the imperfect full detector, a FIFO depth of five is required in order to achieve 100% throughput. At the same time, the FIFO can be seen to be working as expected.

4.2 An improved full detector

All the same, it would be interesting to see how much more expensive a 'perfect' full detector would be compared to the one implemented above. The design of such is completely analogous to that of the perfect empty detector; referring to Table 4.1, it must detect the $N-1$ situation. To accomplish this, the read pointer is first synchronised into the write pointer clock domain, and the write pointer token ring is converted to a one-hot encoded signal. It can then be seen that the i'th position indicates a full situation if the i'th bit of the one-hot write pointer is set, and the synchronised read pointer has a transition from 1 to 0 there; see Figure 4.5.[6]

The result of using this full detector can be seen in Figure 4.6, which simulates a FIFO of depth four. Reading is deliberately delayed a few clock cycles to see if the full signal is asserted, which it is after the third write. However, as soon as reading begins, the full signal is deasserted (the read pointer needs to be synchronised, so there's a latency of one clock cycle; the same is true for the empty detector). After this, the throughput is 100%. Thus, the improved full detector offers a much better performance for shallow FIFOs.

Synthesising the FIFO with the improved full detector reveals that it requires 195 flip-flop bits, as seen in Table 4.3, which is actually only two more than with the simple full detector. It uses 175 slices (2%) and 220 four-input LUTs (1%), which is virtually the same as before. This is for a FIFO depth of five, so if the only reason for choosing five in the first place was to achieve 100% throughput, four may be chosen in this case, which would save 38 flip-flop bits and probably some LUTs as well.

The frequency constraint is, however, 164 MHz (6.09 ns), compared to 189 MHz, and the critical path is through the improved full detector. Thus, this FIFO must be clocked a bit slower. Still, when synthesising on a Spartan3E FPGA, the area savings promised by the imperfect full detector do not seem to offer a reasonable trade-off. It should be noted that this is when using the mesochronous adaption, where one of the synchronisation register rows has been removed; in the asynchronous case, this full detector would require an additional five-bit synchronisation register, and for deeper FIFOs, the improved full detector would be relatively more expensive.

[6] Please refer to the file `fullDetectorImproved.vhd` in Appendix A.2.

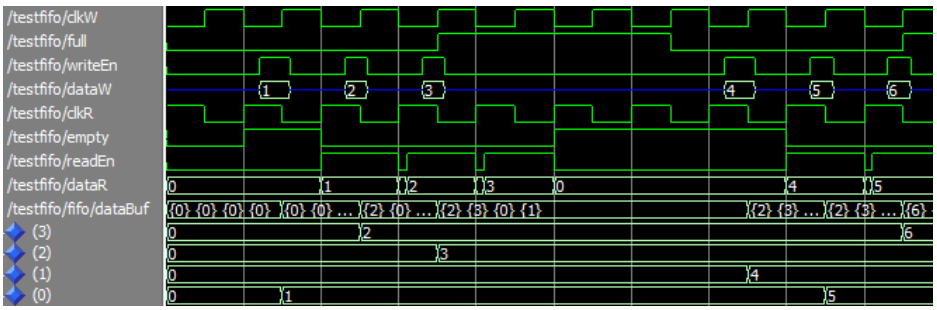

Figure 4.3: FIFO simulation, $N = 4$, 50% throughput

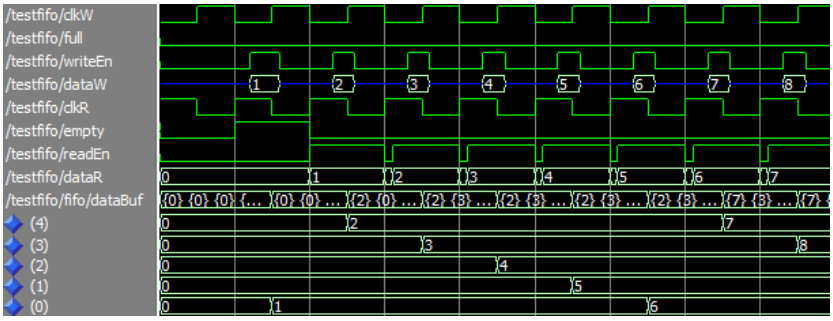

Figure 4.4: FIFO simulation, $N = 5$, 100% throughput

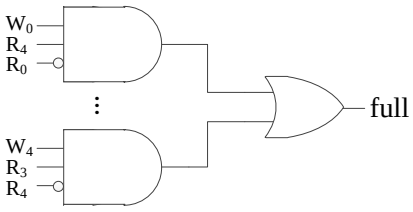

Figure 4.5: Function of the improved full detector

4.3 Clock-gated FIFO synchroniser

Using similar considerations as in Section 3.2, it should be apparent that it would be worthwhile to clock gate the FIFO buffer presented in this chapter. The FIFO is designed so that data is not rotating through the data buffer — rather, the pointers are rotated — which minimises power usage. Still, though, the data registers consume power even when both the write and read enable flags are low.

To mitigate this, it is assumed that an external enable signal is present that indicates whether the FIFO should be active or not (for reasons that will be explained in Chapter 5, the read and write enable signals won't be used for this, and are hard-wired to always high). This signal is synchronous to the write clock domain, which is nice, since the FIFO data buffer also resides in this clock domain. Thus, if the write clock is gated as determined by this enable signal, the data registers, which are the main power drains, will be turned off when the FIFO is not in use. However, power loss will still occur due to the read pointer token ring and the read pointer synchronisation registers.[7]

4.3.1 Synthesis

When synthesising the clock-gated FIFO to the Spartan3E FPGA, the synthesiser reports that it uses 115 slices and 148 four-input LUTs.[8] Since the clock-gated FIFO consists of a wrapper circuit around the non-clock-gated version, which used 213 LUTs, this result cannot be right. Taking into account that clock gating generally does not work directly on FPGAs, this may indicate that the implementation fails already at the synthesis level. To verify this, a post-translate simulation was carried out on the test bench presented in Section 4.3.2; and as expected, the simulation fails with a number of errors about unbound component instances, which indicates that the synthesiser has erroneously 'optimised' away a large part of the circuit. For this reason, the simulation in the following section will be carried out on the behavioural implementation.

The flip-flop utilisation was similar to the non-clock-gated FIFO (Tables 4.2 and 4.3) except that a latch is used in the standard clock-gating cell (Figure 2.1). The flip-flops

[7] Please refer to the file `gatedFifo.vhd` in Appendix A.2 for the VHDL implementation of the clock-gated FIFO.

[8] Please refer to the file `gatedFifo.syr` in Appendix B.2 for the Xilinx XST synthesis report.

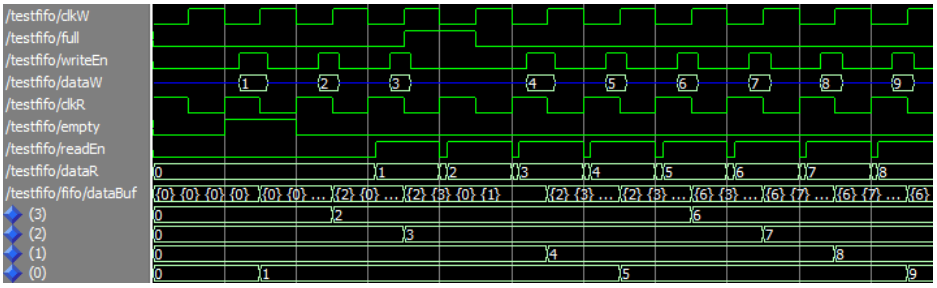

Figure 4.6: FIFO simulation, $N = 4$, 'perfect' full detector

Table 4.3: Register count for FIFO with improved full detector

Description	Count	Bits
5-bit register (token rings)	2	10
5-bit synchronisation register (empty detector)	1	5
5-bit synchronisation register (full detector)	1	5
35-bit data buffers (FIFO)	5	175
	9	195

of the write clock domain are clock gated; that is, the write pointer token ring (5 FFs), the full detector (3 FFs) and the data buffers (175 FFs), for a total of 183 clock-gated flip-flops.

4.3.2 Simulation

The test bench of Section 4.1.4 is modified so that it continuously applies signals to be written to the clock-gated FIFO.[9] These signals consist of sequences of numbers (to account for data), interspersed with zeros (to imitate inactivity); e.g. 0-0-0-0-1-2-3-0-0-0-4-5-6-0-0-0-0…. The enable signal is set to low whenever the input is 0, and high otherwise.

One caveat of only gating the write clock is that, since the write pointer is only rotated when actual data is written (due to the clock gating), while the read pointer is rotated continuously, they may initially become unaligned. Notice the write and read pointers in the bottom of Figure 4.7 during the beginning of the simulation: The write pointer remains constant in its initial position, while the read pointer is rotated five times until it is back at its original position. Put another way, the read pointer does not point to the same address as the write pointer until after four clock periods (counting from when the reset signal is no longer applied), after which the empty signal goes high, which internally prevents further reading. The yellow cursor in Figure 4.7 marks this position. So a correct result cannot be read before this time.

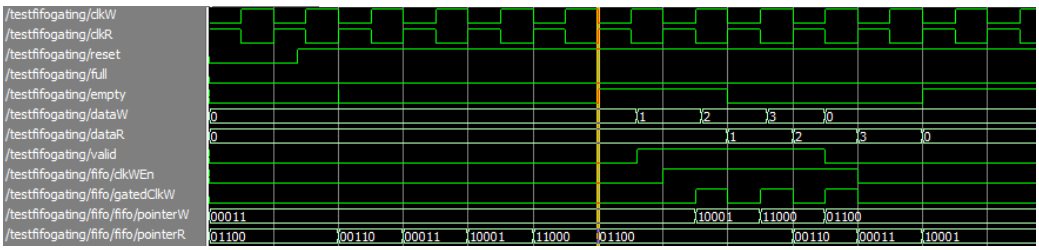

Figure 4.7: Clock-gated FIFO, initial write delay of five clock cycles

Figure 4.8 illustrates this point by commencing writing before the read pointer has

[9] Please refer to the file `testFifo_gating.vhd` in Appendix A.2.

been fully rotated. Since the read pointer does not reach the position written until after four clock cycles, reading cannot start until then. The yellow cursor marks the same position as in Figure 4.7. Also, because the non-optimal full detector detects the 'quasi-full' condition, writing is stalled after two elements have been written, which in turn causes the read sequence to be interrupted after the second element. However, after this initial confusion, which can be prevented by waiting at least four cycles before starting to write data to the FIFO, the clock-gated FIFO behaves as the simple one. Figure 4.9 shows a simulation similar as in Figure 4.8, but using the improved full detector of Section 4.2; this allows all three initial elements to be written without an interruption. This figure also shows the behaviour once the FIFO is operating steadily, where the latency is one period plus the clock phase difference as in the non-clock-gated FIFO buffer.

For the above reasons, to give the read pointer time to attain the correct position, it is recommended to wait at least four clock cycles after initialisation before starting to produce data.

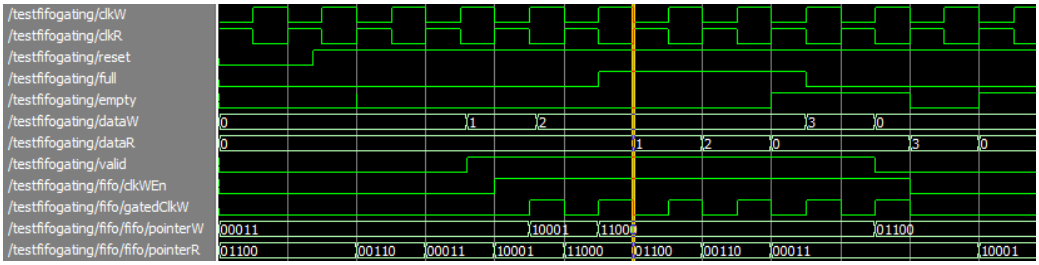

Figure 4.8: Clock-gated FIFO, write delay of two clock cycles, non-optimal full detector

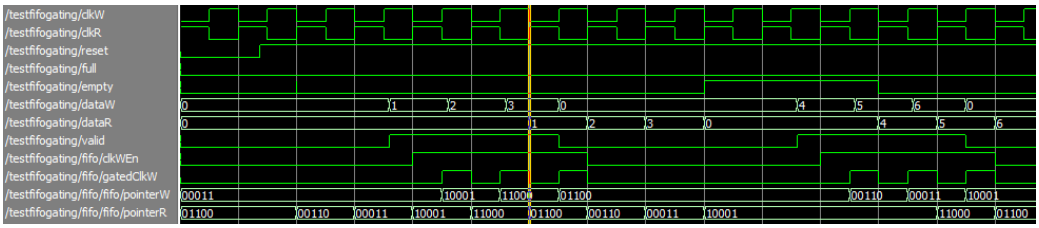

Figure 4.9: Clock-gated FIFO, write delay of two clock cycles, optimal full detector

4.4 Results

In this chapter, a FIFO buffer was implemented on the basis of [MPG07] that can be used for synchronisation between two mesochronous clock domains. Furthermore, an improved was full detector proposed in order to improve throughput, making a 100% throughput possible for FIFOs of depth four instead of five, which was originally required.

This FIFO was clock gated, and the effect of this was tested by simulation. It should be noted that the clock-gated FIFO requires a global initialisation of four clock cycles before it can process data. The results obtained in this chapter are summarised in Table 4.4.

Table 4.4: The results obtained for the FIFO buffer

Free running			Clock gated		
LUTs	Flip-flops	Frequency	LUTs	Flip-flops	Frequency
213	193	189 MHz	n/a	193	n/a

Chapter 5

The Mesochronous Network

This chapter details the analysis and design of a mesochronous network-on-chip router based on the components designed in the previous chapters. First, FIFO buffers will be connected to the inputs of a synchronous router, resulting in a mesochronous router that allows a constant phase difference between the read and write clocks; then, it will be analysed how this approach can be modified to allow the phase difference to slowly drift in a so-called plesiochronous system; and finally, the mesochronous router will be clock gated in order to minimise power consumption when it is not in use.

5.1 Mesochronous router

Using the building blocks introduced in the previous chapters, a mesochronous router can be designed by connecting FIFO buffers to the inputs of the synchronous router, as depicted in Figure 5.1.[1] This ensures the presence of a FIFO between all the router links, enabling synchronisation of data despite a constant clock phase difference between neighbouring routers having the same clock frequency — that is, a mesochronous network. If the FIFO depth is chosen accordingly, the phase difference may even be allowed to slowly drift.

In Figure 5.1, a FIFO buffer is also placed between the router and the local IP core. For simplicity, it is assumed that this is similar to the four other FIFOs; but as mentioned in Chapter 4, the FIFOs used have been simplified to synchronise only in the mesochronous case. Generally, it would probably be desired to clock the IP core independently of the NoC, in which case an asynchronous FIFO should be used. This would require an extra row of synchronisation registers, as in Figure 4.1; otherwise, this FIFO would be similar to the others.

The FIFOs, when connected to the router inputs, are intended to facilitate a continuous flow of data; and when no flit is actually being routed, the crossbar select signal generated by the HPU will ensure that the crossbar simply outputs a flit consisting of logical 0's. For this reason, the read and write enable signals of the FIFO should be constantly high, making the FIFO behave somewhat like a pipeline register. Hence, the full and empty signals are of minor importance and should, during normal operation, never go high; if one of them does go high, this would indicate an abnormal error condition (in a plesiochronous system, this could happen if clock skew caused data to be produced gradually faster, and consumed gradually slower, filling the FIFO up; or vice versa).

[1] Please refer to the file `routerFifo.vhd` in Appendix A.3 for the VHDL implementation of the mesochronous router.

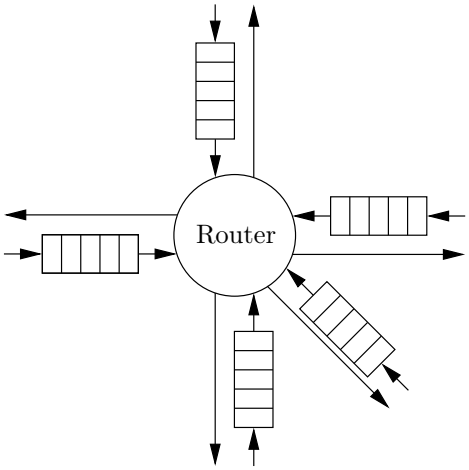

Figure 5.1: A router with its FIFO synchronisers

[MPG07] mentions that, to avoid metastability in the synchronisers of Figure 4.1 when using the FIFO buffer in a mesochronous configuration, the phase difference between the clock signals should be between 90° and 270°. Since we do not expect the empty and full signals to change (as discussed above, they are expected to always be negative), this constraint does not need to be rigorously enforced. All the same, it provides a useful guideline, and we shall in the following assume that neighbouring routers have a clock phase difference of 180°. This would mean that the network is clocked in a check-like pattern. For this reason, and for simplicity, the test bench implicitly assumes that all neighbouring nodes have the same phase difference, so they are represented by the same clock signal. In a real implementation, they could be a few degrees out of phase, and each FIFO buffer would need to use a separate write clock. This would clutter the VHDL code and simulation results somewhat, but would not be a major design change.

Except for the FIFOs connected to the input ports, the router presented in this section is similar to that of Chapter 3.

5.1.1 Synthesis

Because of the large FIFO buffers, the area requirements of the mesochronous router are expected to be considerable. Indeed, the synthesis report shows that, apart from using 1355 flip-flop bits as shown in Table 5.1, it uses 1994 four-input LUTs (11%) and 1450 slices, which is 16% of the total available and four times as many as the synchronous router.[2]

The maximum frequency is 132 MHz with the critical path running from the FIFO buffer to the HPU, where the select signal for the crossbar is generated. This indicates it would probably be worthwhile to put a pipeline register between the FIFO and the HPU, if one could spare an additional 175 flip-flops. It should be noted that this pipeline stage is needed not because of the data, which is effectively pipelined in the FIFO's data buffer, but because of the empty signal, which is needed to determine whether the read pointer can be incremented.

5.1.2 Simulation

The router is tested using an approach similar to that of the synchronous test bench in Section 3.1.5. As with the FIFO buffers in Section 4.1.4, two clock signals are generated with a 180° phase difference, corresponding to the local and neighbouring clocks.[3]

[2] Please refer to the file `routerFifo.syr` in Appendix B.3 for the Xilinx XST synthesis report.

[3] The VHDL implementation of this test bench is available in the file `testRouter_fifo.vhd` in Appendix A.3.

Table 5.1: Register count for the mesochronous router

Description	Count	Bits
35-bit pipeline register (data)	10	350
20-bit pipeline register (select signal)	1	20
4-bit address register (HPU)	5	20
5×35 bi-synchronous FIFO buffer (193 bits)	5	965
	21	1355

In a process triggered on each rising flank of the write clock (simulating a neighbouring router), each FIFO input port is in turn supplied with a new flit according to a predefined test vector stipulating which packages to be sent at which state in the test. Similarly, in a process triggered on the rising flank of the read clock (simulating the local router), the output of the router is read, and the data is compared to the test vector. A warning is generated if an unexpected flit arrives, if no flit arrives when one is expected, or if the sequence number doesn't match.

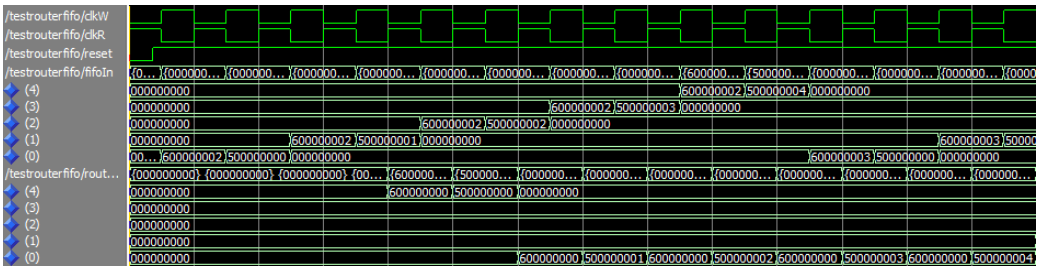

Figure 5.2: Simulation of the mesochronous router

The situation of Figure 5.2 is similar to the synchronous situation of Figure 3.5, where a package is sent to port 0 (bottom of the picture) from all input ports (middle of the picture). The latency can be seen to be three and a half clock periods; two from the router, and one and a half (actually, one plus the phase difference) from the FIFO buffers.

5.1.3 Power consumption

The mesochronous router consists of 1355 flip-flops. Thus, when subjecting it to the same usage scenario as the synchronous router, the flip-flops account for a toggle count of 13550 (see Table 5.2).

Table 5.2: Power estimation of the mesochronous router

Time slot	1	2	3	4	5	6	7	8	9	10
1st pkg	0–3 (start)	0–3 (data)	0–3 (end)							
2nd pkg		1–4 (start)	1–4 (data)	1–4 (end)						
3rd pkg	1–0 (start)	1–0 (data)	1–0 (end)							
Flip-flops	1355	1355	1355	1355	1355	1355	1355	1355	1355	1355

5.2 Plesiochronous considerations

So far, the FIFO buffers used in the mesochronous router have had a depth of five for the somewhat arbitrary reason that this is the minimum depth at which 100% throughput

can be achieved when using the non-optimised full detector. The FIFO depth is, however, interesting because it determines how much clock skew can be tolerated in a plesiochronous system. If, for example, the read clock slowly drifts, gradually increasing the time between writes and reads, at one point the FIFO will become full, and either the throughput will decline, or data will be lost. If the clocks drift toward each other, data will be read too fast, and the FIFO will at some point become empty, which also decreases the throughput. Clock drift is not unrealistic and can happen for various reasons; significantly, wire propagation delay increases with temperature, so if different parts of a system have an uneven load, their temperatures are likely to vary, and the propagation delays will not be constant.

The question then is, how large should the FIFOs be in order to be able to tolerate clock drift? We consider a system containing a FIFO, whose drifting read clock is initially delayed by half a period compared to the write clock, and where the FIFO starts out as empty. At this point, up to two elements need to be stored at the same time, since the FIFO latency is at least one clock period (an element is read 1.5 periods after it has been written). When the read clock has drifted half a period, the two clocks are completely aligned, and the latency through the FIFO is effectively two clock periods; two elements are stored in the FIFO at a time. After another period, the latency is effectively three clock periods, and so the FIFO needs to accommodate three elements, and so forth. This is illustrated in Figure 5.3.

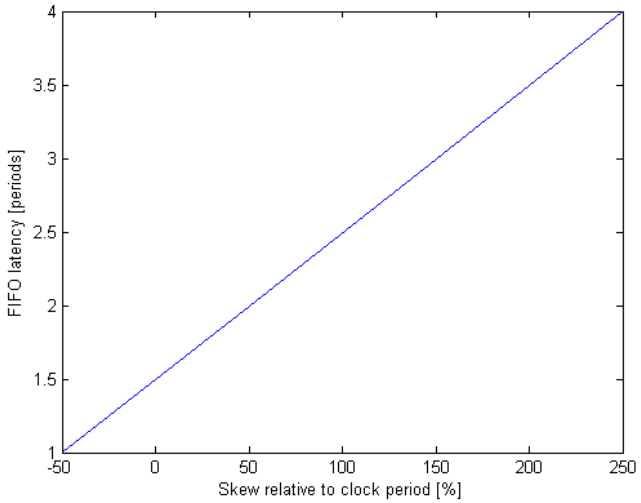

Figure 5.3: FIFO latency as a function of read clock skew

The above can be easily verified in a simulation test environment. A test bench is made where the clock period is 100 ns, but the read clock is delayed by 1 ns every ten clock cycles.[4] Every time the write clock ticks, a flit is sent through input port 0: either a header flit destined for port 2, or a data (and stop) flit containing a unique sequence number. Similarly, every time the read clock ticks, data is read at port 2, and if it is not the proper header flit, or a data flit with the right sequence number, an error is reported. In the ModelSim wave window, signals representing the two clocks, the clock skew and the number of elements currently in the FIFO data buffer (which is the difference between the write and read pointers) are monitored. At the start of the simulation, it can be seen that the number of elements currently stored in the FIFO varies between one and two, and each number accounts for 50% of the time. As the drift increases, the time intervals where two elements are stored increase relative to those where only one element is stored, and when the drift reaches 50 ns (corresponding to a 360° phase difference), two elements are stored in the FIFO all the time.

[4] Please refer to the file `testPleso.vhd` in Appendix A.3.

Figure 5.4 shows the simulation of a FIFO with four elements after a 50 ns clock drift (making the total phase difference 360°). This FIFO uses the improved full detector of Section 4.2. The skew is showed along with the number of elements in the FIFO and the read and write pointers. After the drift reaches 50 ns, the full signal is asserted, which means that the FIFO internally disables the write enable signal. Thus, an element is not written, which causes the test to fail. Evidently, this happens already after a 50 ns clock drift, even though Figure 5.3 predicts that this requires between two and three elements, which a four-element FIFO should be amply suited for. The wave window also shows that the number of elements doesn't even exceed two. So why is the full signal asserted? The problem is that the read pointer needs to be synchronised to the write clock domain, which incurs a delay of one clock period; so when the full detector compares the current write pointer with the previous read pointer, it sees the $N-1$ situation of Table 4.1 and rightly indicates a full situation (remember that the FIFO is designed so that a four-element FIFO can only contain three elements). When the FIFO is full, no more elements can be written, and since the data producer doesn't act on this, data is lost. When using the original, non-improved full detector and a FIFO of depth five, the same happens after a 50 ns skew; so the extra element does not in this way allow for a larger amount of clock skew.

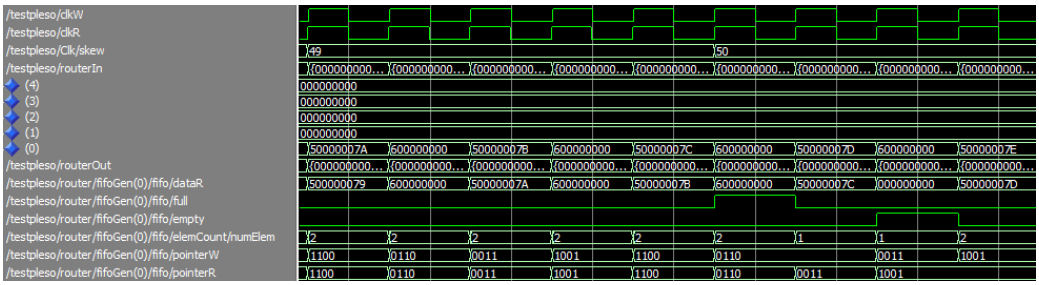

Figure 5.4: Simulation of a plesiochronous system

Note that the FIFO is actually *not* full at this time, so if it allowed a producer to write to it even though the full signal were asserted, that should not present a problem. If not, an extra FIFO element needs to be available compared to what is shown in Figure 5.3; so for example, to allow for a skew of 250%, a six-element FIFO is needed.

In the above, a system has been considered in which reading is slowed, causing the FIFO to fill up. Of course, the same concepts apply if the opposite happens, only the FIFO would need to have a tolerance against buffer underflow. For this reason, it might be advisable to initialise the system in such a way that all FIFOs are about half full, which would provide a sort of elasticity in both directions. Of course, a buffer underflow may not be as serious as an overflow if the receiver is designed to ignore empty/invalid flits even when they arrive in the middle of a package; but it degrades throughput, which may break a real-time guarantee in a guaranteed service layer.

5.3 Clock-gated mesochronous router

Since the mesochronous router is simply a synchronous router connected with FIFO buffers at the input ports, a clock-gated version can be obtained by clock gating the individual components as described in the previous chapters; a clock-gated mesochronous router then simply consists of the clock-gated synchronous router of Section 3.2 with the clock-gated FIFOs of Section 4.3 at its input ports. This will ensure that, when all input ports to a mesochronous router are inactive, the whole router along with the write clock domain of the FIFO buffers will be turned off. If a single port becomes active, only that particular FIFO will be turned on, but the whole router will have to be activated. This is not perfect, but as explained earlier, it is not trivial to fine-tune clock gating of the router due to the interlinked pipeline signals and the crossbar. It should also be noted that the five FIFO

buffers account for 965 flip-flop bits (see Table 3.7), while the router itself only uses 390; that is, 71% of the flip-flop utilisation lies in the FIFOs. For this reason, it makes sense to concentrate on fine-tuning the clock gating of the FIFO buffers. [5]

5.3.1 Synthesis

The register count for the clock-gated mesochronous router is presented in Table 5.3. In addition to this, six latches are used in the standard clock-gating cells.

Since the clock-gated FIFO buffers cannot by synthesised, no LUT count or speed estimation can be given; but since the clock-gating logic itself is pretty simple, the clock-gated mesochronous router should not use substantially more LUTs than the 1994 used by the non-clock-gated version, and it is not expected to be much slower; for a comparison, refer to Table 3.7, where the overhead caused by the clock gating was very slight.

Table 5.3: Register count for the clock-gated mesochronous router

Description	Count	Bits
Clock-gated router	1	392
5×35 clock-gated FIFO buffer (193 bits)	5	965
		1357

5.3.2 Simulation

In Figure 5.5, the clock-gated mesochronous router is simulated using the same test bench as in Section 5.1.2. The test bench determines that the packages are routed to the correct output ports, so the functionality of the router is thus verified. As for the clock gating, please refer to the bottom of the figure showing the gated clocks. In the beginning of the simulation, all the input lines are inactive, so all the gated clocks are turned off. Then a flit is sent to input port 0, causing the write clock of the FIFO at that port to be turned on. After two clock cycles, this is picked up by the router, whose clock is then also turned on. The figure shows how the router clock remains active, while the FIFO write clocks are turned on and off individually.

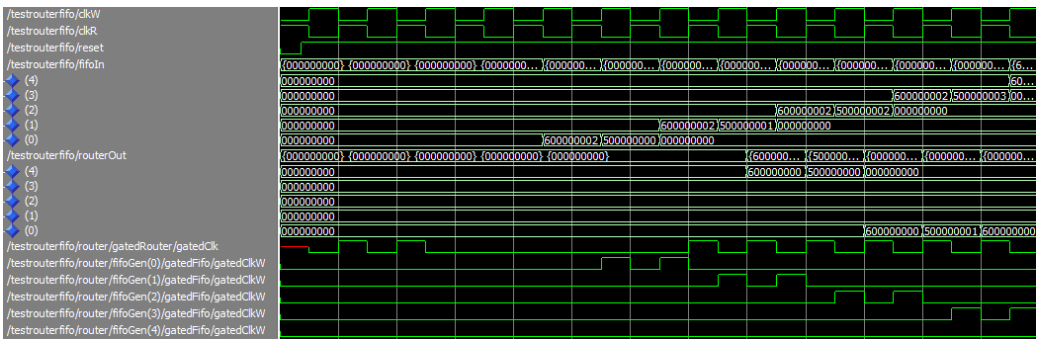

Figure 5.5: Simulation of the clock-gated mesochronous router

5.3.3 Power consumption

Referring to Section 4.3.1, the clock-gated FIFO buffer contains 193 flip-flops and one latch, of which 183 flip-flops are clock gated and disabled whenever that particular input port is inactive. The mesochronous router itself consists of 392 flip-flops and one latch, of which 390 are clock gated. Figure 5.6 shows the gated clock signals for the given usage

[5] Please refer to the file `gatedRouterFifo.vhd` in Appendix A.3 for the VHDL implementation of the clock-gated mesochronous router.

scenario, where the read and write clocks are 180° out of phase. It is worth noticing that the clock signal for the router itself is not turned on until the incoming signal has been processed through the FIFO buffers. The calculation in Table 5.4 shows that the flip-flops account for a toggle count of 4567.

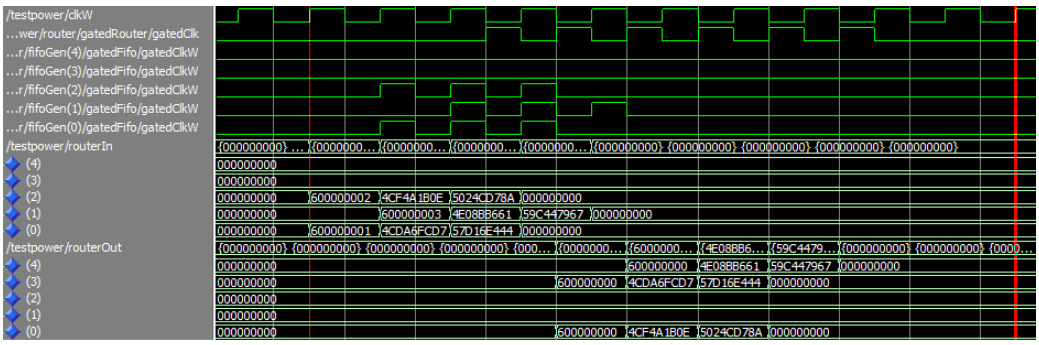

Figure 5.6: Analysis of power consumption for the mesochronous router

Table 5.4: Power estimation of the clock-gated mesochronous router

Time slot	1	2	3	4	5	6	7	8	9	10
1st pkg	0–3 (start)	0–3 (data)	0–3 (end)							
2nd pkg		1–4 (start)	1–4 (data)	1–4 (end)						
3rd pkg	1–0 (start)	1–0 (data)	1–0 (end)							
Flip-flops										
FIFOs (enabled)	0	388	582	582	194	0	0	0	0	0
FIFOs (disabled)	55	33	22	22	44	55	55	55	55	55
Router	3	3	3	393	393	393	393	393	393	3
Total	58	424	607	997	631	448	448	448	448	58

The above number refers to the standard usage scenario, as it was defined in Chapter 3. While this is useful to compare the two router designs and the effect of the clock gating, it would also be interesting to extrapolate the power consumption to other usage percentages. By making a calculation like the one done in Table 5.4 for various usage scenarios, defining the usage percentage as the number of used links divided by the total number of links available (which is 50 for ten time slots), the graph in Figure 5.7 appears. Note that the power consumption is not uniquely defined for a given percentage, as this depends on the exact layout of the flits, so this is somewhat arbitrary. Still, the graph shows an almost linear dependence for most of the spectrum, except for very low percentages. A likely explanation for this is the granularity of the router clock gating, which requires the whole router to be turned on for multiple clock cycles to route just one flit; of course the penalty for doing this is smaller, the more flits are routed. This can be seen by the fact that the graph gradually approaches the linear function between the minimum and maximum power consumptions (which are $58 \cdot 10 = 580$ and $(194 \cdot 5 + 393) \cdot 10$, respectively). The message is, not unexpectedly, that for small usage percentages, a penalty is paid in power consumption for the router clock-gating approach; for higher percentages, this doesn't matter. When measuring compared to this straight line, it can also be concluded that for a usage of 18%, there's a power 'over-head' of 56% compared to a perfectly clock-gated router.

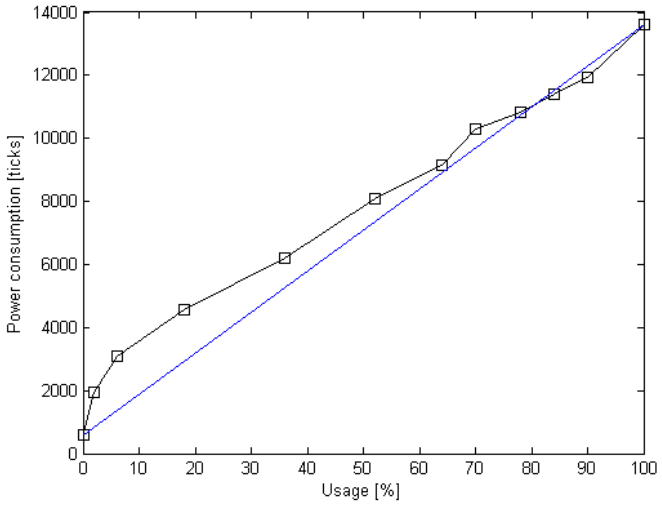

Figure 5.7: Power consumption as a function of usage percentage

Table 5.5: The results obtained for the mesochronous router

Free running				Clock gated			
LUTs	Flip-flops	Power	Frequency	LUTs	Flip-flops	Power	Frequency
1994	1355	13550	132 MHz	n/a	1357	4567	n/a

5.4 Results

In this chapter, the FIFO buffers presented in Chapter 4 were combined with the synchronous router of Chapter 3 to create a mesochronous router, of which a clock-gated version was then proposed. These were analysed with regard to area cost and tested by simulating them in ModelSim. An estimate of their power consumptions was then conducted, and the mesochronous router was compared to an ideally clock-gated router. The results obtained are summarised in Table 5.5.

Also, a plesiochronous system was considered, in which the ramifications of a slowly varying clock phase difference between neighbouring routers were examined. Figure 5.3 can be used as a guideline when choosing how deep the FIFOs should be in order to tolerate a certain drift, but the behaviour of the full detector means that the FIFO should be an element deeper than indicated in the figure.

Chapter 6

FPGA Implementation and Test

This chapter describes a synthesisable test bench for the mesochronous router implemented in the previous chapter. This is used to verify that the router works not only when simulated, but also when run on an FPGA. The point is to provide an indication that the design works in practise — a sort of 'proof of concept' — and not to design an actual network-on-chip. Thus, the FPGA implementation presented here does not in itself constitute a useful system, other than as a confirmation of the functionality of the router.

The first part of this chapter will discuss the design of the test bench itself, after which the test bench will be simulated and synthesised, and the test results will be briefly discussed.

6.1 Test bench design

To test the functionality of the mesochronous router on an FPGA, a test bench inspired by the one used in Section 5.1.2 will be used: During the test, packages should be sent through all possible routes. The challenge is to design the test bench so as to be able to verify that this happens correctly; to do this, the test bench keeps track of how many packages are sent, and how many are received, at each output port. Furthermore, each data flit is given a unique code so as to be able to verify that it arrives at the correct destination. This test is not exhaustive — for example, while it does check that flits arrive in the correct order, it makes no assumptions about the latency through the router — but used along with the ModelSim simulation results, it gives a pretty good indication that the router is working as intended. However, it should be noted that if the MTBF for metastability is high enough — even in the presence of potential design errors in the synchronisers — these errors would probably not be caught by this manual testing method; instead, a much more rigorous mechanism should be used (e.g. running the test millions of times). Again, it is emphasised that the test bench is intended as a proof-of-concept, to demonstrate a working design, and not as an industry-standard stress test.[1]

In order to be able to check the contents and order of arriving flits, a FIFO buffer is maintained for each output port. When the sender sends a flit through the router, the same flit is written to the appropriate FIFO; and likewise, the receiver compares the output of the router with the next output of its FIFO. An alternative approach would be to hard-code the test vectors into the receiver, which would work equally well, but not

[1] For the VHDL implementation of this test bench, refer to the file `fpgaTest.vhd` in Appendix A.4.

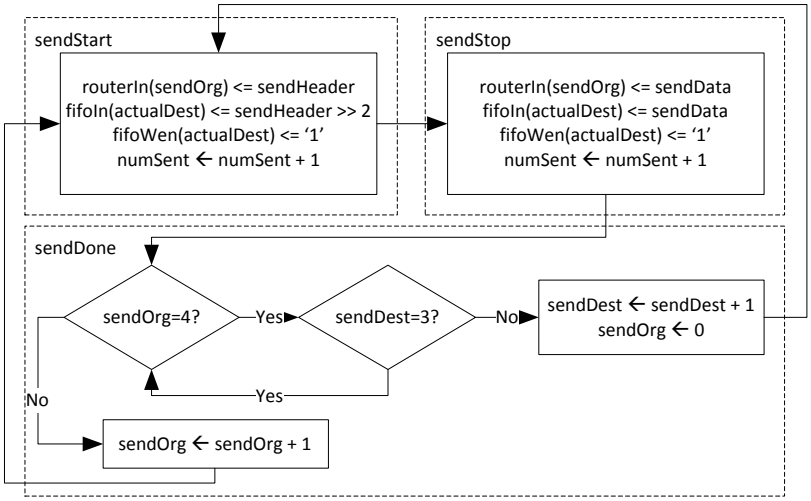

Figure 6.1: ASM chart of send state machine

offer the same degree of flexibility; and as an added bonus, the FIFOs are tested even more thoroughly this way.

Figure 6.1 shows the ASM chart of the send state machine, which is in the write clock domain. Two counters keep track of what to do: `sendOrg` maintains the origin, or router input port, from which the next flit should be sent; and `sendDest` stores the next flit destination. For each value $0 \ldots 4$ of `sendOrg`, `sendDest` cycles through the values $0 \ldots 3$, so that all combinations of origin and destination are reached (output port 4 is reached when `sendDest` equals `sendOrg`). A decoder (not shown in the figure) sets `sendHeader` to the appropriate header flit to contain the address information given by `sendDest`. The signal `actualDest` is set to the actual destination, which is `sendDest` unless `sendOrg` = `sendDest`, in which case the actual destination is the local port (4); this signal is only used in order to write to the correct FIFO buffer. One caveat is that, as per the design of the HPU, the address field of the first flit is rotated in the router, so that the first two bits always contain the next address; for this reason, the same operation is applied to all start of package flits before these are written to the FIFO.

In the send state machine, the first two states — `sendStart` and `sendStop` — send out the start of package and end of package flits, respectively. `sendDone` calculates the next values of `sendOrg` and `sendDest`, as shown in Figure 6.1. For simplicity, only two flits are sent per package instead of three.

Figure 6.2 depicts the receive state machine in the read clock domain. This state machine is repeated in the test bench, so that each output port of the router is monitored by its own independent state machine. For this reason, all the signals mentioned in the following have a width of five, and each state machine operates on its own element in these arrays.

In the `idle` state, the router output ports are checked to see if they contain actual data; if this is the case, the state machine has a transition to the `recvStart` state, otherwise it remains in `idle`. Thus, the data (if any) received in `idle` is the first flit of the package, so the receive state machine is always one flit behind; the `recvBuf` register is used to remember the last received flit. In the `recvStart` state, the last received flit (`recvBuf`) is compared to the output of the FIFO buffer. If these match, a counter is incremented; each output port has its own counter to keep track of how many flits have been correctly received at that output port. If they don't match, an error counter is incremented, which contains the total number of errors. The same thing is done in the `recvStop` state.

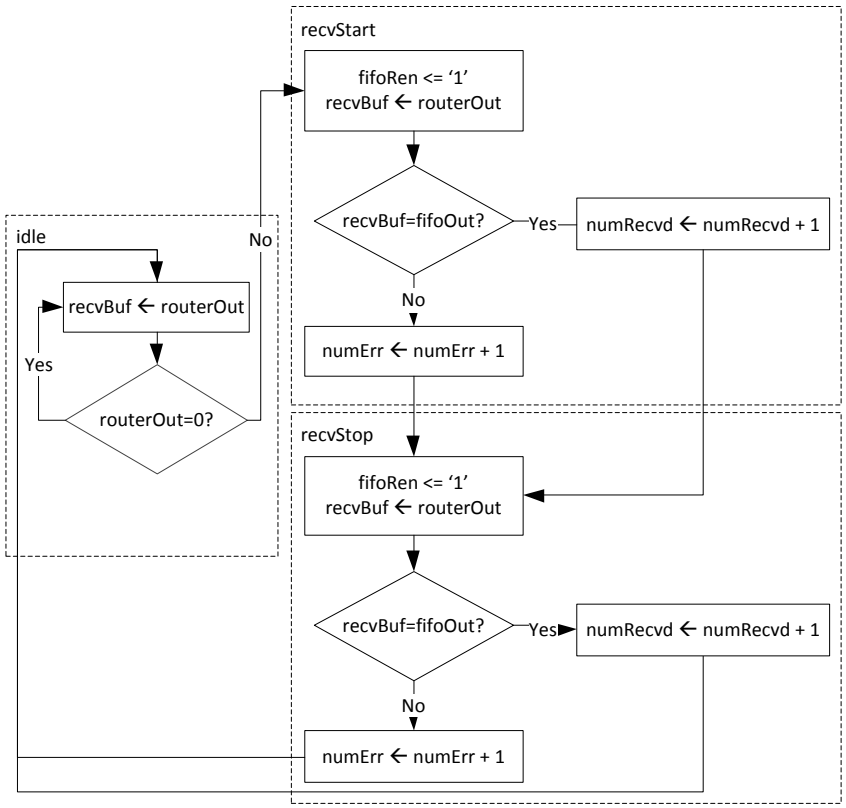

Figure 6.2: ASM chart of receive state machine

To verify that the test has completed successfully, the seven-segment display[2] on the Nexys2 board is used along with the switches. Two of the four digits always show the register containing the number of sent flits; and using the switches, the remaining two digits can be selected to display one of the six other counters (number of received flits at each output port, and number of errors).

The circuit is clocked using the on-board clock generator with a frequency of 50 MHz, which is wired to the write clock. The read clock is set to the inverse, making a 180° clock phase difference between the two clock domains.

6.2 Simulation

The test environment is simulated using ModelSim to ensure that it works correctly before it is run on the FPGA.[3] In particular, the contents of the registers for the number of sent and received flits are inspected.

In the sender circuit, two flits are sent to each output port from each of the four other input ports. Thus, a total of eight flits should arrive at each of the five ports, and 40 flits should be sent in total. Figure 6.3 shows a simulation of the test bench when the last flit is being received by the receiver at output port 3. It can be seen that 40 (28 hex) flits have indeed been sent, and eight have been received at each port. `numRecvd`[5] contains the number of errors, which is initialised to the value 16 (10 hex) in order to be able

[2] To manipulate this display, the module written by *JWC*, downloaded from http://blog.jwcxz.com/?p=647, is used.

[3] To simulate the test bench, the wrapper in `fpgaTestSim.vhd` of Appendix A.4 is used.

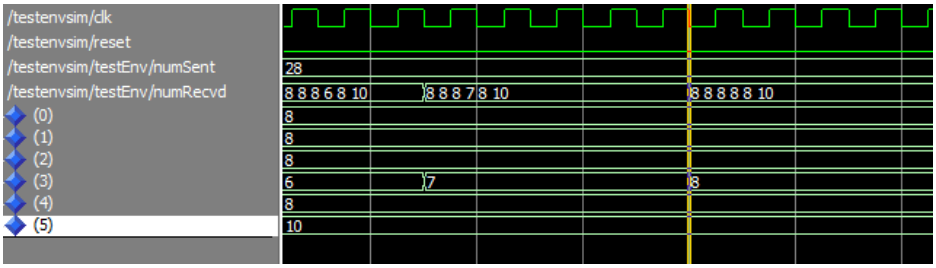

Figure 6.3: Simulation of the synthesisable test bench

to differentiate it from nothing when viewing it in the on-board display; thus, the figure shows that no errors have occurred.

It can also be seen in the figure that the receive registers are written on the falling edge of the clock. Notice that there are five clock cycles between each write operation; this is because the internal clock is slowed by a factor five using a DCM (see Section 6.3).

6.3 Synthesis

No attempt has been made to optimise the test bench code using such techniques as operator or functionality sharing. As expected, the synthesiser gives a number of warnings, advising that some comparators and arithmetic circuits could be shared in order to reduce area. It also warns that a number of signals are constantly zero, so they have been trimmed; this is no cause for concern and is caused by the fact that the test bench doesn't use all of the bits of the flits. All in all, the test bench uses 4095 LUTs (23%), 3493 flip-flops (20%) and 2937 slices (33%).[4]

The critical path is through the logic in the receive state machines, which features a comparator and several adder circuits, and has a minimum propagation delay of 11.2 ns. However, the receive logic reads from the test FIFO and compares this to what was actually received on the router output, and since data is read from this FIFO on the rising clock edge, while the receive logic is 180° out of phase with this, so that it must write to its own registers on the falling edge, it effectively only has half a clock period to perform this operation. Thus, the minimum clock period is 22.4 ns, corresponding to a maximum frequency of 44.6 MHz. Since the on-board crystal is 50 MHz, a Digital Clock Manager (DCM) is instantiated in order to generate a slower clock signal (10 MHz).

6.4 Results

When synthesising the circuit without a DCM to slow the clock, the functionality is sporadic, which is to be expected: Setup times are frequently violated, causing the logic to fail. However, when using a clock of 10 MHz, the test has never been observed to fail.

Using the switches to display the values of the status registers, it can be verified that the router performs correctly: The number of sent flits is 40, and the number of flits received at each port is eight; and the number of errors is none. Thus, the synthesisable test bench confirms that the mesochronous router implementation works in practise on an FPGA.

[4] Please refer to the file `TestEnv.syr` in Appendix B.4 for the Xilinx XST synthesis report.

Chapter 7

Discussion

This chapter presents a discussion of the designs and implementations introduced in the previous chapters. The synchronous and mesochronous routers will be compared with each other as well as with the asynchronous router of [SS11], and their area costs and power consumptions will be discussed. In the second part of this chapter, possible improvements to the current work will proposed and briefly discussed.

7.1 Results

Table 7.1 summarises the results obtained for both the synchronous and mesochronous routers. It is immediately clear that the synchronous router is much smaller, and much faster, than the mesochronous equivalent; however, the figures in the table do not reflect the cost of scaling a synchronous network, where it becomes prohibitively expensive to distribute a non-skewed clock signal for large networks, as discussed in e.g. [HG11, MPCVG08, GH10, HG11]. For this reason, the disadvantages of the synchronous network outweigh the benefits as they appear in Table 7.1.

Table 7.1: The results obtained for the synchronous and mesochronous routers

	Free running				Clock gated			
	LUTs	Flip-flops	Power	Freq.	LUTs	Flip-flops	Power	Freq.
Sync.	761	390	3900	257 MHz	764	392	2370	238 MHz
Meso.	1994	1355	13550	132 MHz	n/a	1357	4567	n/a

Instead, it makes sense to compare the mesochronous network to an asynchronous implementation. [SS11] implements two asynchronous versions of the router presented here, using 1090 and 1475 logic gates (excluding latches, flip-flops and 160 multiplexers, see [SS11, Table I]). If we allow four gates for each of the multiplexers,[1] the two asynchronous routers require 1730 and 2115 gates, respectively. While the LUT count presented in Table 7.1 cannot directly be converted to a gate count, a single four-input LUT can probably, on average, implement the functionality equivalent of 2–4 gates. In the best-case scenario, where a LUT corresponds to two gates, the synchronous router then requires 1522 gates, while the mesochronous router requires 3988 gates. Thus, the synchronous circuit is slightly smaller than the asynchronous circuit, which seems not unreasonable, while the mesochronous circuit is almost twice as large as the asynchronous circuit; and this does

[1] See [BV09, Fig. 2.28]; a one-bit 2-to-1 multiplexer can be implemented with two *and*-gates, one *or*-gate and a *not*-gate.

not even consider the excessive amount of flip-flops used by the mesochronous circuits. The results thus seem to favour an asynchronous implementation.

To put the LUT counts into perspective, the author of this thesis implemented a *MiniMIPS* processor during a project at DTU, which is a fully functional MIPS, but with only a limited set of instructions; this required 4241 LUTs. Also, the Xilinx MicroBlaze CPU uses 1324 LUTs and the PicoBlaze CPU 204.[2] Thus, the size of the router alone — and this does not include other parts of the interconnect, such as the network adaptor — approaches, or even exceeds, that of a fairly advanced IP that could be connected to the network. This emphasises the challenges faced when designing SoCs.

Reverting to the results of Table 7.1, the figures for the power consumption unfortunately do not allow us to compare this with other designs. However, it can be remarked that while the power usage of the clock-gated mesochronous router is about twice as large as that of the synchronous router for the 20% usage scenario, this does not seem excessive when considering the many synchronisers required for mesochronous operation. As Figure 5.7 shows, the clock gating of the individual FIFO buffers causes the power consumption to depend almost linearly on the load for all but the smallest percentages, for which the power consumed by the router itself becomes significant. Whether effort should be put into clock gating the router further depends a great deal on the exact usage scenario; if the router processes only a few flits most of the time, it would probably be worth it, but if the packages arrive in bursts, interspersed by complete inactivity, it shouldn't matter much.

7.2 Further work

While a working, practical implementation of both the synchronous and mesochronous routers has been presented in the preceding chapters, there are several areas in which the designs could be further optimised. In this section, ways to improve clock gating, area costs and measurements are proposed.

7.2.1 Clock gating

As mentioned, whether or not effort should be invested in a more fine-tuned clock gating depends on the usage scenario; but if it is deemed necessary, it is possible (albeit non-trivial) to further save power in two ways. First, the clock gating of the router itself could offer a better granularity, so that each pipeline register is turned on individually, both before and after the crossbar (see Figure 3.2). The problem is that the registers traversed after the crossbar depend on the address information of the header flit; so the most obvious way to implement this would be for the HPU on the incoming port to generate an enable signal for the relevant register on the outgoing port. Each of the five HPUs would thus have to be able to enable each of the five outgoing pipeline registers, in addition to the registers in front of the crossbar. This is unlikely to be cheap in terms of area.

Second, in the current implementation, only the write clock domain of the FIFO buffers is clock gated. To clock gate the read domain (consisting of the read pointer and some synchronisation registers), the enable signal would have to be synchronised across this transition, which would delay it at least one clock cycle. To implement a working FIFO while doing this is likely to be tricky.

7.2.2 Area costs

To minimise the area costs of the routers is another aspect on which further work could focus. As mentioned in Section 3.1.4, the crossbar alone requires 525 four-input LUTs. The crossbar is implemented after the design of Figure 3.3, which consists of 20 two-input *and* gates and five four-input *or* gates, for 35 bits; so to get a total of 525 LUTs means that one LUT can implement the functionality of two two-input *and* gates and one

[2] Obtained from http://www.1-core.com/library/digital/soft-cpu-cores/.

four-input *or* gate, since $(20/2 + 5/1) \cdot 35 = 525$. It has to be assumed that this is the best the synthesiser can do, and it does have the advantage of being a reasonably fast implementation; it could probably be done in more layers with fewer LUTs, but this would be slower.

Another expensive part is the multiplexer on the output port of the FIFO that selects between the data registers, which requires close to 200 LUTs as mentioned in Section 4.1.3. If tri-state buffers are available on the target architecture (this is not the case for the Spartan3E FPGA), this could be implemented a lot cheaper as originally proposed in [MPG07]. Even if this reduces the area of each FIFO by the equivalent of only 100 LUTs, this optimisation alone would mean a 25% area reduction for the mesochronous router.

However, the most obvious way to minimise area costs is to dimension the FIFOs appropriately. The original, non-improved full detector requires a FIFO depth of five while providing the same amount of clock skew tolerance as a four-element FIFO with the improved full detector. If an element is 35 bits long, and if each router requires five FIFOs, this means a reduction of 175 flip-flop bits per router in the data buffer alone (the token rings and synchronisation registers have to be considered as well). Furthermore, Section 5.2 shows that both full detectors (and thus, probably the empty detector as well) positively inhibit clock skew tolerance, preventing the FIFOs to be used to their fullest extent. Thus, it should be considered whether the full and empty detectors could be completely removed from the FIFO when it is used in a mesochronous router like the one presented here; during normal operation, the FIFO should *never* be full or empty anyway.

7.2.3 Measuring power and area

The somewhat cumbersome arguments presented above touch upon a relevant limitation of the results presented in this thesis: Since area costs are measured using LUTs (and flip-flops) used on a particular FPGA, and power consumption is measured using low-to-high flip-flop clock ticks for an arbitrary usage scenario, it is very difficult to compare the design to other implementations. In other words, it would be nice to have the circuit laid out, which would make it possible to derive the exact number of standard cells or transistors along with the exact wattage required to process different number of packages. This is not a trivial process, and it requires relevant experience of using CAD tools such as Synopsis, which the author of this thesis regrettably lacks, and the attainment of which is outside the scope of this thesis. Furthermore, to get a meaningful result, the complexity and power consumption of the other network components, such as the network adaptors, would also have to be considered. These are tasks that have to be completed if the routers presented here are to be used in actual designs.

Chapter 8

Conclusion

In this thesis, a mesochronous network-on-chip router has been presented. Its area costs and power consumption have been analysed, and its functionality has been verified using simulation and with a proof-of-concept implementation on an FPGA. The results show that while a working implementation has been achieved, it comes at the price of relatively high area costs compared to a similar, asynchronous router. Specifically, while the mesochronous router is almost three times as large as a simple synchronous router when comparing LUTs, it is also almost twice as large as an asynchronous router.

During the work with the mesochronous router, a bi-synchronous FIFO buffer used for synchronisation, based on a design by [MPG07], has also been studied and analysed. It turned out to be nontrivial to incorporate it into the design, particularly because of the full detector, of which a new one has been designed and implemented in the course of this thesis. However, when analysing the tolerance for clock skew of the mesochronous router in a plesiochronous system, it turned out that the full (and empty) detector reduces the tolerance, so it may be considered to remove it altogether from the FIFOs.

All in all, a working mesochronous NoC router has been designed, although the disadvantages in terms of die area and speed induced by the mesochronous design paradigm severely puts into question the practicality of the solution.

Bibliography

[AJI07] Khalil Arshak, Essa Jafer, and Christian Ibala. Power testing of an FPGA based system using ModelSim code coverage capability. In *2007 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 157–160, Los Alamitos, CA, 2007. IEEE Computer Society.

[Aro12] Mohit Arora. *The Art of Hardware Architecture*. Springer, New York, NY, 2012.

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1), 2006.

[BS05] Tobias Bjerregaard and Jens Sparsø. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1226–1231, Los Alamitos, CA, 2005. IEEE Computer Society.

[BV09] Stephen Brown and Zvonko Vranesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill, third edition, 2009.

[DP98] William J. Dally and John W. Poulton. *Digital Systems Engineering*. Cambridge University Press, Cambridge, UK, 1998.

[DT04] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[DYN03] José Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks, an Engineering Approach*. Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[GH10] Kees Goossens and Andreas Hansson. The Aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Design Automation Conference*, pages 306–311, Anaheim, CA, 2010. ACM.

[Gin11] Ran Ginosar. Metastability and synchronizers: A tutorial. *IEEE Design and Test of Computers*, 28(5):23–35, 2011.

[HG11] Andreas Hansson and Kees Goossens. *On-Chip Interconnect with Aelite*. Springer, New York, NY, 2011.

[MPCVG08] Ivan Miró Panades, Fabien Clermidy, Pascal Vivet, and Alain Greiner. Physical implementation of the DSPIN network-on-chip in the FAUST architecture. In *Second ACM/IEEE International Symposium on Networks-on-Chip*, pages 139–148, Los Alamitos, CA, 2008. IEEE Computer Society.

[MPG07] Ivan Miró Panades and Alain Greiner. Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in GALS architectures. In *First International Symposium on Networks-on-Chip*, pages 83–94, Los Alamitos, CA, 2007. IEEE Computer Society.

[MPGS06] Ivan Miró Panades, Alain Greiner, and Abbas Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the GALS approach. In *First International Conference on Nanonetworks and Workshops*, pages 1–5, Los Alamitos, CA, 2006. IEEE Computer Society.

[SS11] Rasmus Bo Sørensen and Jens Sparsø. Circuit design of a router for an asynchronous TDM network-on-chip. Preprint, personal communication, 2011.

[Xil11] Xilinx, Inc. Spartan-3 Generation FPGA User Guide. Available at http://www.xilinx.com/support/documentation/user_guides/ug331.pdf, 2011.

Appendix A

Code listings

A.1 The Synchronous Network

vhdl/router.vhd

```
-- router.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Synchronous NoC router. Consists of HPUs, crossbar and pipeline registers.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity router is
  port ( clk:  in std_logic;
       reset:  in std_logic;
      inPort: in XbarPort;
      outPort: out XbarPort
     );
end router;

architecture struct of router is
  signal sel0, sel1, sel2, sel3, sel4: std_logic_vector(3 downto 0);

  -- pipeline registers
  signal XbarSel, XbarSelNext:  std_logic_vector(19 downto 0);
  signal XbarOut, XbarOutNext:  XbarPort;
  signal HPUout, HPUoutNext:    XbarPort;
begin
  port0:  entity work.HPU
      port map(clk=>clk, reset=>reset, inLine=>inPort(0), outLine=>HPUoutNext(0),
           sel=>sel0);
  port1:  entity work.HPU
      port map(clk=>clk, reset=>reset, inLine=>inPort(1), outLine=>HPUoutNext(1),
           sel=>sel1);
  port2:  entity work.HPU
      port map(clk=>clk, reset=>reset, inLine=>inPort(2), outLine=>HPUoutNext(2),
```

```
            sel=>sel2);
  port3:  entity work.HPU
      port map(clk=>clk, reset=>reset, inLine=>inPort(3), outLine=>HPUoutNext(3),
            sel=>sel3);
  port4:  entity work.HPU
      port map(clk=>clk, reset=>reset, inLine=>inPort(4), outLine=>HPUoutNext(4),
            sel=>sel4);

  XbarSelNext <= sel4 & sel3 & sel2 & sel1 & sel0;

  xbar: entity work.Xbar
      port map(func=>XbarSel, inPort=>HPUout, outPort=>XbarOutNext);

  outPort <= XbarOut;

  process(clk, reset)
  begin
    if reset = '0' then
      XbarSel <= (others => '0');
      XbarOut <= (others => (others => '0' ));
      HPUout <= (others => (others => '0' ));
    elsif clk'event and clk = '1' then
      XbarSel <= XbarSelNext;
      XbarOut <= XbarOutNext;
      HPUout <= HPUoutNext;
    end if;
  end process;
end struct;
```

vhdl/xbar.vhd

```
-- xbar.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Crossbar for the NoC router.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity Xbar is
  port(
      func: in std_logic_vector(19 downto 0);
      inPort: in XbarPort;
      outPort: out XbarPort
    );
end Xbar;

-- Func format:
--  source port:  4    3    2    1    0
--  dest port:  1032 1042 1034 4032 1432

architecture structure of Xbar is
  signal sel0, sel1, sel2, sel3, sel4:  std_logic_vector(3 downto 0);
begin
  sel0 <= func(3 downto 0);
  sel1 <= func(7 downto 4);
  sel2 <= func(11 downto 8);
  sel3 <= func(15 downto 12);
  sel4 <= func(19 downto 16);

  outPort(0) <= (inPort(1) and (dataLine'range=>sel1(2))) or
          (inPort(2) and (dataLine'range=>sel2(2))) or
          (inPort(3) and (dataLine'range=>sel3(2))) or
          (inPort(4) and (dataLine'range=>sel4(2)));
  outPort(1) <= (inPort(0) and (dataLine'range=>sel0(3))) or
          (inPort(2) and (dataLine'range=>sel2(3))) or
          (inPort(3) and (dataLine'range=>sel3(3))) or
          (inPort(4) and (dataLine'range=>sel4(3)));
  outPort(2) <= (inPort(0) and (dataLine'range=>sel0(0))) or
          (inPort(1) and (dataLine'range=>sel1(0))) or
          (inPort(3) and (dataLine'range=>sel3(0))) or
          (inPort(4) and (dataLine'range=>sel4(0)));
  outPort(3) <= (inPort(0) and (dataLine'range=>sel0(1))) or
          (inPort(1) and (dataLine'range=>sel1(1))) or
          (inPort(2) and (dataLine'range=>sel2(1))) or
          (inPort(4) and (dataLine'range=>sel4(1)));
  outPort(4) <= (inPort(0) and (dataLine'range=>sel0(2))) or
          (inPort(1) and (dataLine'range=>sel1(3))) or
          (inPort(2) and (dataLine'range=>sel2(0))) or
          (inPort(3) and (dataLine'range=>sel3(1)));

end structure;
```

vhdl/hpu.vhd

```
-- hpu.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Header parsing unit for the NoC router. See [thesis, Fig. 3.4].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity HPU is
  port(
      clk:  in std_logic;
      reset:  in std_logic;
      inLine: in dataLine;
      outLine: out dataLine;
      sel:  out std_logic_vector(3 downto 0)
    );
end HPU;

architecture struct of HPU is
  signal SOP:   std_logic;
  signal EOP:   std_logic;
  signal dest:  std_logic_vector(1 downto 0);

  signal selInt, selIntNext:  std_logic_vector(3 downto 0);
  signal decodedSel:      std_logic_vector(3 downto 0);
  signal outInt:  dataLine;
```

```
begin
  SOP <= inLine(33);
  EOP <= inLine(32);
  dest <= inLine(1 downto 0);
  outLine <= outInt;

  -- binary decoder, dest field into a one-hot signal
  decodedSel(0) <= '1' when dest = "00" else '0';
  decodedSel(1) <= '1' when dest = "01" else '0';
  decodedSel(2) <= '1' when dest = "10" else '0';
  decodedSel(3) <= '1' when dest = "11" else '0';

  selIntNext <= decodedSel when SOP = '1' else (selInt and (selInt'range=>not(EOP
      )));

  sel <= selInt when EOP = '1' else selIntNext;
  outInt <= "11000" & inLine(31 downto 2) when SOP = '1' else inLine;

  process (reset,clk)
  begin
    if reset = '0' then
      selInt <= (others => '0');
    elsif clk'event and clk = '1' then
      selInt <= selIntNext;
    end if;
  end process;

end struct;
```

vhdl/testRouter.vhd

```
-- testRouter.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Test bench for a synchronous NoC router.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;
use work.txt_util.all;

entity testRouter is
end testRouter;

architecture behaviour of testRouter is
  signal clk:          std_logic := '0';
  signal reset:        std_logic;
  signal routerIn, routerOut: XbarPort; -- 0 is SOUTH, 1 is WEST, 2 is NORTH, 3
      is EAST, 4 is LOCAL

  -- test vectors
  constant OUT_NORTH: dataLine := "11000000000000000000000000000000000";
  constant OUT_EAST: dataLine := "11000000000000000000000000000000001";
  constant OUT_SOUTH: dataLine := "11000000000000000000000000000000010";
  constant OUT_WEST: dataLine := "11000000000000000000000000000000011";
  constant FLIT_STOP: dataLine := "10100000000000000000000000000000000";

  constant TEST_LENGTH: integer := 7;
  type testVectorType is array(0 to TEST_LENGTH-1) of dataLine;
  type outNumType is array(0 to TEST_LENGTH-1) of integer;
  constant TEST_VECTOR: testVectorType := (OUT_SOUTH, OUT_WEST, LINE_ZERO,
      LINE_ZERO, LINE_ZERO, OUT_NORTH, OUT_EAST);
  constant OUT_NUM: outNumType := (0, 1, 0, 0, 0, 2, 3);  -- ports at which the
      above flits are expected to arrive

begin
  reset <= '0', '1' after 37 ns;
  clk <= not clk after 50 ns;

  router: entity work.gatedRouter
    port map(clk=>clk, reset=>reset, inPort=>routerIn, outPort=>routerOut);

  wBehaviour: process is
    variable outPort: integer;
  begin
    routerIn <= (others=>(others=>'0'));
    wait until reset = '1' and clk'event and clk = '1';

    for idx in 0 to TEST_LENGTH-1 loop
      report "Writing with idx := " & str(idx) severity note;
      for i in 0 to 4 loop
        -- apply test input
        routerIn <= (others=>LINE_ZERO);
        routerIn(i) <= TEST_VECTOR(idx);
        wait until clk'event and clk = '1';
        if TEST_VECTOR(idx) /= LINE_ZERO then
          routerIn(i) <= FLIT_STOP or std_logic_vector(to_unsigned(i, 35));
        else
          routerIn(i) <= LINE_ZERO;
        end if;
        wait until clk'event and clk = '1';
      end loop;
    end loop;
    routerIn <= (others=>LINE_ZERO);
    wait until reset = '1';
  end process wBehaviour;

  rBehaviour: process is
    variable outPort: integer;
  begin
    wait until reset = '1' and clk'event and clk = '1';
    -- two period latency due to pipeline in router
    wait until clk'event and clk = '1';
    wait until clk'event and clk = '1';

    for idx in 0 to TEST_LENGTH-1 loop
      report "Reading with outNum := " & str(OUT_NUM(idx)) severity note;
      for i in 0 to 4 loop
        -- check for correct output
        wait for 10 ns;
        if OUT_NUM(idx) = i then
          outPort := 4; -- local output
        else
          outPort := OUT_NUM(idx);
        end if;
        if routerOut(outPort) /= (TEST_VECTOR(idx)(34 downto 2) & "00") then
          report "Output mismatch header flit. idx := " & str(idx) & ", i := " &
              str(i) & ", outPort := " & str(outPort) severity error;
        end if;
        wait until clk'event and clk = '1';
        wait for 10 ns;
```

```
        if routerOut(outPort) /= (FLIT_STOP or std_logic_vector(to_unsigned(i,
            35))) and TEST_VECTOR(idx) /= LINE_ZERO then
          report "Output mismatch stop flit. idx := " & str(idx) & ", i := " &
              str(i) & ", outPort := " & str(outPort) severity error;
        end if;
        wait until clk'event and clk = '1';
      end loop;
    end loop;
```

vhdl/testPower.vhd

```
-- testPower.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Test bench to generate a 'typical' load scenario for power estimation.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;
use work.txt_util.all;

entity testPower is
end testPower;

architecture behaviour of testPower is
  signal clkW:              std_logic := '0';
  signal clkR:              std_logic := '1';
  signal reset:             std_logic;
  signal routerIn, routerOut:     XbarPort; -- 0 is SOUTH, 1 is WEST, 2 is NORTH,
       3 is EAST, 4 is LOCAL

  -- test vectors
  constant OUT_NORTH: dataLine  := "11000000000000000000000000000000000000";
  constant OUT_EAST: dataLine   := "11000000000000000000000000000000000001";
  constant OUT_SOUTH: dataLine  := "11000000000000000000000000000000000010";
  constant OUT_WEST: dataLine   := "11000000000000000000000000000000000011";
  constant FLIT_STOP: dataLine  := "10100000000000000000000000000000000000";
  constant RAND1: dataLine     := "10011001101101001101111110011010111"; -- from /
      dev/random
  constant RAND2: dataLine     := "10001111101000101101110010001000100";
  constant RAND3: dataLine     := "10011100000100010111011011001100001";
  constant RAND4: dataLine     := "10010011100010001000111001011001111";
  constant RAND5: dataLine     := "10011001111010010100001101100001110";
  constant RAND6: dataLine     := "10100000010010011001101011110001010";

begin
  reset <= '0', '1' after 37 ns;
  clkW <= not clkW after 50 ns;
  clkR <= not clkR after 50 ns;

  router:
  entity work.gatedrouterFifo
    --port map(clk=>clkW, reset=>reset, inPort=>routerIn, outPort=>routerOut);
    port map(clkLocal=>clkR, clkNeighbour=>clkW, reset=>reset, inPort=>routerIn,
        outPort=>routerOut);

  wBehaviour: process is
  begin
    routerIn <= (others=>(others=>'0'));
    wait until reset = '1' and clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    report "CONGRATULATIONS! If no failures, then all tests completed
        successfully!" severity note;
    wait until reset = '1';
  end process rBehaviour;

end behaviour;
```

```
    wait until clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';

    report "Simulation start" severity note;

    -- time slot 1
    routerIn(0) <= OUT_EAST;
    routerIn(2) <= OUT_SOUTH;
    wait until clkW'event and clkW = '1';

    -- time slot 2
    routerIn(0) <= RAND1;
    routerIn(1) <= OUT_WEST;
    routerIn(2) <= RAND5;
    wait until clkW'event and clkW = '1';

    -- time slot 3
    routerIn(0) <= RAND2 or FLIT_STOP;
    routerIn(1) <= RAND3;
    routerIn(2) <= RAND6 or FLIT_STOP;
    wait until clkW'event and clkW = '1';

    -- time slot 4
    routerIn(0) <= LINE_ZERO;
    routerIn(1) <= RAND4 or FLIT_STOP;
    routerIn(2) <= LINE_ZERO;
    wait until clkW'event and clkW = '1';

    -- time slot 5
    routerIn(1) <= LINE_ZERO;
    wait until clkW'event and clkW = '1';

    -- time slot 6
    wait until clkW'event and clkW = '1';

    -- time slot 7
    wait until clkW'event and clkW = '1';

    -- time slot 8
    wait until clkW'event and clkW = '1';

    -- time slot 9
    wait until clkW'event and clkW = '1';

    -- time slot 10
    wait until clkW'event and clkW = '1';

    report "Simulation done!" severity note;
    wait until reset'event;

  end process wBehaviour;

end behaviour;
```

vhdl/gatedRouter.vhd

```
-- gatedRouter.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- NoC router with clock gating.
-- See clock-gating cell in [Arora, Fig. 2.26].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity gatedRouter is
  port ( clk:    in  std_logic;
         reset: in  std_logic;
         inPort:  in  XbarPort;
         outPort: out XbarPort
       );
end gatedRouter;

architecture structure of gatedRouter is
  signal validSigIn:              std_logic;
  signal validSigOut1, validSigOut1Next:  std_logic;
  signal validSigOut2, validSigOut2Next:  std_logic;
  signal gateEnable, clkEn, gatedClk:    std_logic;
begin
  process(clk, reset)
  begin
    if reset = '0' then
      validSigOut1 <= '1';
      validSigOut2 <= '1';
    elsif clk'event and clk = '1' then
      validSigOut1 <= validSigOut1Next;
      validSigOut2 <= validSigOut2Next;
    end if;
  end process;

  -- clock gating strategy: turn everything off when no valid input signals
  validSigIn <= inPort(0)(34) or inPort(1)(34) or inPort(2)(34) or inPort(3)(34)
      or inPort(4)(34);
  validSigOut1Next <= validSigIn;          -- DFF, allow for latency of 2 periods
  validSigOut2Next <= validSigOut1;          -- through router before output is
       available
  gateEnable <= validSigIn or validSigOut2;
  clkEn <= gateEnable when clk = '0' else clkEn;  -- latch, see [Fig. 2.26]
  gatedClk <= clkEn and clk;

  router: entity work.router
    port map(clk=>gatedClk, reset=>reset, inPort=>inPort, outPort=>outPort);
end structure;
```

vhdl/types.vhd

```
-- types.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Definition of data types.

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

package types is

  subtype dataLine is std_logic_vector(34 downto 0);
  type  XbarPort is array(4 downto 0) of dataLine;

  constant LINE_ZERO: dataLine := (others => '0');
end types;

package body types is

end types;
```

A.2 A FIFO Synchroniser for Mesochronous Networks

vhdl/fifo.vhd

```
-- fifo.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- FIFO synchroniser for mesochronous router.
-- See [Miro Panades & Greiner, 2007].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--use work.txt_util.all;  -- provides log2 for numElem, see below

entity fifo is
  generic(
      N:    integer := 5;
      W:    integer := 35
       );
  port( clkW: in std_logic;
      clkR: in std_logic;
      reset: in std_logic;
      writeEn: in std_logic;
      readEn: in std_logic;
      dataW: in std_logic_vector (W-1 downto 0);
      dataR: out std_logic_vector (W-1 downto 0);
      full: out std_logic;
      empty: out std_logic
    );
end fifo;

architecture structure of fifo is
  signal writeEnInt, readEnInt: std_logic;
  signal pointerW, pointerR:    std_logic_vector(N-1 downto 0); -- write and read
       pointers
  signal andPointR:         std_logic_vector(N-1 downto 0); -- ANDed read pointer
  signal writeIndex, readIndex: std_logic_vector(N-1 downto 0); -- one-hot
      encoded index into dataBuf
  signal fullInt, emptyInt:   std_logic;

  type BufferType is array (N-1 downto 0) of std_logic_vector(W-1 downto 0);
  signal dataBuf, dataBufNext:  BufferType;
begin
  ------------------------
  -- write pointer module
  writeEnInt <= writeEn and not fullInt;
  writeP: entity work.tokenRing
      generic map (N=>N, default=>3)                    -- LSBs are "...0011"
      port map (clk=>clkW, en=>writeEnInt, reset=>reset, data=>pointerW);

  writeIndex <= pointerW and (pointerW(0) & pointerW(N-1 downto 1));      -- AND
      with neighbouring bits [Fig. 7]

  -- write to the dataBuf register specified by the one-hot encoded writeIndex
  -- simply write to register i if the i'th bit is set and write is enabled
  dataBufWriteGen:
  for i in 0 to N-1 generate
    dataBufNext(i) <= dataW when (writeIndex(i) and writeEnInt) = '1' else
        dataBuf(i);
end generate;

------------------------
-- read pointer module
readEnInt <= readEn and not emptyInt;
readP:  entity work.tokenRing
     generic map (N=>N, default=>12)                    -- LSBs are "...1100"
     port map (clk=>clkR, en=>readEnInt, reset=>reset, data=>pointerR);

andPointR <= pointerR and (pointerR(0) & pointerR(N-1 downto 1));      -- AND
     with neighbouring bits
readIndex <= andPointR(1 downto 0) & andPointR(N-1 downto 2);      -- rotate two
      bits to align with dataBuf [Fig. 7]

-- read signal multiplexer - decode the one-hot encoded readIndex into the
     appropriate dataBuf signal
-- read from register i if the i'th bit is set, see [Fig. 7], and read is
     enabled
process(readIndex,dataBuf,readEnInt)
begin
  dataR <= (others=>'0');
  for i in 0 to N-1 loop
    if (readIndex(i) and readEnInt) = '1' then
      dataR <= dataBuf(i);
    end if;
  end loop;
end process;

------------------------
-- full and empty detectors
fullDet: entity work.fullDetectorImproved
     generic map (N=>N)
     --port map (clk=>clkW, reset=>reset, writeEn=>writeEn, writeP=>pointerW,
          readP=>pointerR, full=>fullInt);
     port map (clk=>clkW, reset=>reset, writeP=>pointerW, readP=>pointerR, full
          =>fullInt);

emptyDet: entity work.emptyDetector
     generic map (N=>N)
     port map (clk=>clkR, reset=>reset, writeP=>pointerW, readP=>pointerR, empty
          =>emptyInt);

full <= fullInt;
empty <= emptyInt;

------------------------
-- register process
regProc: process(clkW, reset)
begin
  if reset = '0' then
    dataBuf <= (others => (others => '0'));
  elsif clkW'event and clkW = '1' then
    dataBuf <= dataBufNext;
  end if;
end process regProc;

-- TEST: the following provides a local variable, numElem,
```

```
  --      containing the number of elements currently in the FIFO.
  --      Shouldn't be synthesised, but useful for simulation.
  -- elemCount: process
    -- variable shiftAmount, readP: integer;
    -- variable numElem: integer;
  -- begin
    -- shiftAmount := log2(to_integer(unsigned(writeIndex)));
    -- readP := to_integer(rotate_right(unsigned(readIndex), shiftAmount));
    -- numElem := log2(readP);
    -- wait on writeIndex, readIndex;
  -- end process;
end structure;
```

vhdl/tokenring.vhd

```
-- tokenring.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Token ring for the read/write pointers in the FIFO buffers.
-- See [Miro Panades & Greiner, 2007]

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tokenRing is
  generic(
      N:       natural := 5; -- size of token ring
      default:  natural := 1  -- default value of token ring
       );
  port(
      clk:     in std_logic;
      en:      in std_logic;
      reset:     in std_logic;
      data:    out std_logic_vector(N-1 downto 0)
    );
end tokenRing;

architecture behaviour of tokenRing is
  signal ring, ringNext:  std_logic_vector(N-1 downto 0);
begin
  data <= ring;
  -- if enabled, rotate the token one place right
  ringNext <= ring(0) & ring(N-1 downto 1) when en = '1' else ring;

  process(clk, reset)
  begin
    if reset = '0' then
      ring <= std_logic_vector(to_unsigned(default, N));
    elsif clk'event and clk = '1' then
      ring <= ringNext;
    end if;
  end process;
end behaviour;
```

vhdl/fullDetector.vhd

```
-- fulldetector.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Full detector for the FIFO synchroniser. See [Miro Panades & Greiner, 2007].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fullDetector is
  generic (
      N:     integer := 5          -- depth of FIFO
      );
  port (  clk:  in std_logic;
      reset:  in std_logic;
      writeEn: in std_logic;
      writeP: in std_logic_vector(N-1 downto 0);
      readP:  in std_logic_vector(N-1 downto 0);
      full: out std_logic
     );
end fullDetector;

architecture behaviour of fullDetector is
  -- synchronisation flipflops
  signal sync0, sync0Next: std_logic;
  signal sync1, sync1Next: std_logic;
  signal sync2, sync2Next: std_logic;
  signal andSig:       std_logic_vector(N-1 downto 0);
  signal orSig:      std_logic;
  signal fullS:      std_logic;

  constant ZEROS:      std_logic_vector(N-1 downto 0) := (others => '0');
begin

  andSig <= writeP and readP;
  orSig <= '0' when andSig = ZEROS else '1';
  sync0Next <= orSig;
  sync1Next <= sync0;
  fullS <= sync1;

  -- optimisation, see [Miro Panades et al, fig. 9]
  sync2Next <= fullS or writeEn;
  full <= sync2 and fullS;

  regProc: process(clk, reset)
  begin
    if reset = '0' then
      sync0 <= '0';
      sync1 <= '0';
      sync2 <= '0';
    elsif clk'event and clk = '1' then
      sync0 <= sync0Next;
      sync1 <= sync1Next;
      sync2 <= sync2Next;
    end if;
  end process regProc;
end behaviour;
```

vhdl/emptyDetector.vhd

```
-- emptydetector.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Implements an empty detector for the FIFO.
-- See [Miro Panades & Greiner, 2007].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity emptyDetector is
  generic(
      N: natural := 5
     );
  port( clk:  in std_logic;
        reset:  in std_logic;
        writeP: in std_logic_vector(N-1 downto 0);
        readP:  in std_logic_vector(N-1 downto 0);
        empty:  out std_logic
    );
end emptyDetector;

architecture structure of emptyDetector is
  -- synchronisation flip-flops
  signal syncWriteP, syncWritePNext: std_logic_vector(N-1 downto 0);  --
       synchronised write pointer
  signal rotReadP:    std_logic_vector(N-1 downto 0);       -- rotated read
       pointer
  signal rotSyncWriteP: std_logic_vector(N-1 downto 0);       -- rotated
       synchronised write pointer
  signal andReadP:    std_logic_vector(N-1 downto 0);       -- ANDed read pointer
  signal andSig:      std_logic_vector(N-1 downto 0);       -- ANDed read and
       write pointers

  constant ZEROS:     std_logic_vector(N-1 downto 0) := (others => '0');
begin
  syncWritePNext <= writeP;

  rotReadP <= readP(0) & readP(N-1 downto 1);           -- rotate one bit right
  andReadP <= readP and rotReadP;                     -- AND with neighbouring bits

  rotSyncWriteP <= syncWriteP(N-2 downto 0) & syncWriteP(N-1);  -- rotate one bit
       left
  andSig <= not syncWriteP and rotSyncWriteP and andReadP;    -- AND with
       neighbouring bits and read pointer

  empty <= '0' when andSig = ZEROS else '1';            -- OR the result

  regProc: process(clk, reset)
  begin
    if reset = '0' then
      syncWriteP <= (others => '0');
    elsif clk'event and clk = '1' then
      syncWriteP <= syncWritePNext;
    end if;
  end process regProc;
end structure;
```

vhdl/testFifo.vhd

```
-- testfifo.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Test bench for the FIFO buffer.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testFifo is
end testFifo;

architecture behaviour of testFifo is
  signal clkW: std_logic := '0';
  signal clkR: std_logic := '1';
  signal reset: std_logic;

  signal writeEn, readEn, full, empty: std_logic;
  signal dataW, dataR: std_logic_vector(3 downto 0);

begin
  reset <= '0', '1' after 37 ns;
  clkW <= not clkW after 50 ns;
  clkR <= not clkR after 50 ns;

  dataW <= (others => 'Z');

  fifo: entity work.fifo
    generic map(N=>5, W=>dataW'length)
    port map(clkW=>clkW, clkR=>clkR, reset=>reset, writeEn=>writeEn, readEn=>
         readEn, dataW=>dataW, dataR=>dataR, full=>full, empty=>empty);

  wBehaviour: process is
    variable count: integer := 1;
  begin
    writeEn <= '0';
    dataW <= (others => 'Z');
    wait until reset = '1' and clkW'event and clkW = '1';

    while count <= 15 loop
      wait for 70 ns;
      if full = '1' then
        report "waiting to write to own FIFO..." severity note;
        wait until full = '0';
      else
        dataW <= std_logic_vector(to_unsigned(count, dataW'length));
        writeEn <= '1';
        wait until clkW'event and clkW = '1';
        wait for 10 ns;
        writeEn <= '0';
        dataW <= (others => 'Z');
        count := count + 1;
      end if;
    end loop;
    wait until empty = '1';
  end process wBehaviour;

  rBehaviour: process is
```

```
    variable count: integer := 1;
  begin
    readEn <= '0';
    wait until reset = '1' and clkR'event and clkR = '1';
    --report "waiting a little before reading from FIFO..." severity note;
    --wait until clkR'event and clkR = '1';
    --wait until clkR'event and clkR = '1';
    while count <= 15 loop
      wait for 10 ns;
      if empty = '1' then
        report "waiting to read from FIFO..." severity note;
        wait until empty = '0';
      end if;
      readEn <= '1';
      wait until clkR'event and clkR = '1';
      readEn <= '0';
      if not dataR = std_logic_vector(to_unsigned(count, dataR'length)) then
        report "whoa, read something unexpected from own FIFO" severity warning;
      end if;
      count := count + 1;
    end loop;
  end process rBehaviour;
end behaviour;
```

vhdl/fullDetectorImproved.vhd

```
-- emptydetector.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Improved full detector for the FIFO synchroniser (inverse of original empty
    detector).
-- See [Miro Panades & Greiner, 2007].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fullDetectorImproved is
  generic(
      N: natural
      );
  port( clk:  in std_logic; -- write clock
       reset:  in std_logic;
       writeP: in std_logic_vector(N-1 downto 0);
       readP:  in std_logic_vector(N-1 downto 0);
       full: out std_logic
    );
end fullDetectorImproved;

architecture structure of fullDetectorImproved is
  -- synchronisation flipflops
  signal syncReadP, syncReadPNext: std_logic_vector(N-1 downto 0); --
      synchronised read pointer
  signal rotSyncReadP:  std_logic_vector(N-1 downto 0);        -- rotated
      synchronised read pointer
  signal andWriteP:    std_logic_vector(N-1 downto 0);        -- ANDed write
      pointer
  signal rotWriteP:    std_logic_vector(N-1 downto 0);
  signal andSig:       std_logic_vector(N-1 downto 0);        -- ANDed read and
      write pointers

  constant ZEROS:      std_logic_vector(N-1 downto 0) := (others => '0');
begin
  syncReadPNext <= readP;

  andWriteP <= writeP and (writeP(0) & writeP(N-1 downto 1));   -- one-hot encode
      the write pointer
  rotWriteP <= andWriteP(N-2 downto 0) & andWriteP(N-1);       -- rotate one bit
      left

  rotSyncReadP <= syncReadP(N-2 downto 0) & syncReadP(N-1);    -- rotate one bit
      left
  andSig <= syncReadP and not rotSyncReadP and rotWriteP;      -- AND with
      neighbouring bits and write pointer

  full <= '0' when andSig = ZEROS else '1';            -- OR the result

  regProc: process(clk, reset)
  begin
    if reset = '0' then
      syncReadP <= (others => '0');
    elsif clk'event and clk = '1' then
      syncReadP <= syncReadPNext;
    end if;
  end process regProc;
end structure;
```

vhdl/gatedFifo.vhd

```
-- gatedFifo.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- FIFO buffer with clock gating.
-- Write and read enable are implicitly turned on whenever the 'enable' signal is
    high.
-- That is, when 'enable' is high, the producer is expected to continuously
    present data
-- on the input, and the consumer is expected to read data on the output.
-- For gating theory, [Arora, Fig. 2.26].

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gatedFifo is
  generic(
      N:    integer := 5;
      W:    integer := 35
      );
  port( clkW: in std_logic;
       clkR: in std_logic;
       reset:  in std_logic;
       enable: in std_logic;        -- enable signal for clock gating
       dataW:  in std_logic_vector (W-1 downto 0);
       dataR:  out std_logic_vector (W-1 downto 0);
```

```
      full: out std_logic;
      empty:  out std_logic
    );
end gatedFifo;

architecture structure of gatedFifo is
  signal clkWEn, gatedClkW: std_logic;
begin
  clkWEn <= enable when clkW = '0' else clkWEn; -- latch
  gatedClkW <= clkW and clkWEn;

  fifo:
  entity work.fifo
    generic map(N=>N, W=>W)
    port map(clkW=>gatedclkW, clkR=>clkR, reset=>reset, writeEn=>'1',
      readEn=>'1', dataW=>dataW, dataR=>dataR, full=>full, empty=>empty);

end structure;
```

vhdl/testFifo_gating.vhd

```
-- testfifo_gating.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Test bench for the clock-gated FIFO buffer.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.txt_util.all;

entity testFifoGating is
end testFifoGating;

architecture behaviour of testFifoGating is
  signal clkW:      std_logic := '0';
  signal clkR:      std_logic := '1';
  signal reset:     std_logic;

  signal full, empty:   std_logic;
  signal dataW, dataR:  std_logic_vector(34 downto 0);
  signal valid:     std_logic;

begin
  reset <= '0', '1' after 137 ns;
  clkW <= not clkW after 50 ns;
  clkR <= not clkR after 50 ns;

  valid <= '0' when dataW = (dataW'range=>'0') else '1';  -- data valid signal

  fifo: entity work.gatedFifo
    --generic map(N=>5, W=>dataW'length)
    port map(clkW=>clkW, clkR=>clkR, reset=>reset, enable=>valid, dataW=>dataW,
        dataR=>dataR, full=>full, empty=>empty);

  wBehaviour: process is
    variable i: integer := 1;
    variable j: integer := 0;
  begin
    dataW <= (others => '0');
    wait until reset = '1' and clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    --wait until clkW'event and clkW = '1';
    --wait until clkW'event and clkW = '1';
    --wait until clkW'event and clkW = '1';

    while i <= 100 loop
      j := 0;
      while j < 3 loop
        wait for 10 ns;
        if full = '1' then
          report "waiting to write to FIFO..." severity note;
          wait until full = '0';
        end if;
        dataW <= std_logic_vector(to_unsigned(i, dataW'length));
        wait until clkW'event and clkW = '1';
        i := i + 1;
        j := j + 1;
      end loop;
      dataW <= (others => '0');
      wait until clkW'event and clkW = '1';
      wait until clkW'event and clkW = '1';
      wait until clkW'event and clkW = '1';
      wait until clkW'event and clkW = '1';
      --wait until clkW'event and clkW = '1';
    end loop;
    wait until empty = '1';
  end process wBehaviour;

  rBehaviour: process is
    variable count: integer := 1;
  begin
    wait until reset = '1' and clkR'event and clkR = '1';
    --wait until clkR'event and clkR = '1';

    while count <= 100 loop
      wait for 10 ns;
      if empty = '1' or (dataR = (dataR'range => '0')) then
        report "waiting to read from FIFO..." severity note;
        wait until clkR'event and clkR = '1';
        next;
      end if;
      --wait until clkR'event and clkR = '1';
      --wait for 10 ns;
      if dataR /= std_logic_vector(to_unsigned(count, dataR'length)) then
        if dataR = std_logic_vector(to_unsigned(count-1, dataR'length)) then
          report "read old signal; clock gating enabled?" severity note;
          wait until clkR'event and clkR = '1';
          next;
        else
          report "whoa, read something unexpected from FIFO, expected count := "
              & str(count) severity error;
        end if;
      else
        report "Read count := " & str(count) & " as expected" severity note;
      end if;
      count := count + 1;
      wait until clkR'event and clkR = '1';
    end loop;
  end process rBehaviour;
end behaviour;
```

A.3 The Mesochronous Network

vhdl/routerFifo.vhd

```
-- routerFifo.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Mesochronous NoC router. Uses FIFOs to synchronise input signals to a standard
      router.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity routerFifo is
  port ( clkLocal:     in  std_logic;
         clkNeighbour: in  std_logic;
         reset:        in  std_logic;
         inPort:       in  XbarPort;
         outPort:      out XbarPort
       );
end routerFifo;

architecture structure of routerFifo is
  signal fifoOut: XbarPort;
  signal fifoFull, fifoEmpty: std_logic_vector(4 downto 0);
begin
  fifoGen:
  for i in 0 to 4 generate
    fifo:
    entity work.fifo
      generic map(N=>5, W=>35)
      port map(clkW=>clkNeighbour, clkR=>clkLocal, reset=>reset, writeEn=>reset,
           readEn=>reset,
        dataW=>inPort(i), dataR=>fifoOut(i), full=>fifoFull(i), empty=>fifoEmpty(i)
            );
  end generate;

  router:
  entity work.router
    port map(clk=>ClkLocal, reset=>reset, inPort=>fifoOut, outPort=>outPort);
end structure;
```

vhdl/testRouter_fifo.vhd

```
-- testrouter_fifo.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Test bench for a mesochronous NoC router.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;
use work.txt_util.all;

entity testRouterFifo is
end testRouterFifo;

architecture behaviour of testRouterFifo is
  signal clkW:              std_logic := '0';
  signal clkR:              std_logic := '1';
  signal reset:             std_logic;
  signal fifoIn, routerOut:     XbarPort; -- 0 is SOUTH, 1 is WEST, 2 is NORTH, 3
       is EAST, 4 is LOCAL

  -- test vectors
  constant OUT_NORTH: dataLine := "11000000000000000000000000000000000000";
  constant OUT_EAST: dataLine := "11000000000000000000000000000000000001";
  constant OUT_SOUTH: dataLine := "11000000000000000000000000000000000010";
  constant OUT_WEST: dataLine := "11000000000000000000000000000000000011";
  constant FLIT_STOP: dataLine := "10100000000000000000000000000000000000";

  constant TEST_LENGTH: integer := 7;
  type testVectorType is array(0 to TEST_LENGTH-1) of dataLine;
  type outNumType is array(0 to TEST_LENGTH-1) of integer;
  constant TEST_VECTOR: testVectorType := (OUT_SOUTH, OUT_WEST, LINE_ZERO,
       LINE_ZERO, LINE_ZERO, OUT_NORTH, OUT_EAST);
  constant OUT_NUM: outNumType := (0, 1, 0, 0, 0, 2, 3);  -- ports at which the
       above flits are expected to arrive

begin
  reset <= '0', '1' after 37 ns;
  clkW <= not clkW after 50 ns;
  clkR <= not clkR after 50 ns;

  router:
  entity work.gatedRouterFifo --gatedRouterFifo
    port map(clkLocal=>clkR, clkNeighbour=>clkW, reset=>reset, inPort=>fifoIn,
         outPort=>routerOut);

  wBehaviour: process is
    variable outPort: integer;
  begin
    fifoIn <= (others=>(others=>'0'));
    wait until reset = '1' and clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';
    wait until clkW'event and clkW = '1';

    for idx in 0 to TEST_LENGTH-1 loop
      report "Writing with idx := " & str(idx) severity note;
      for i in 0 to 4 loop
        -- apply test input
        fifoIn <= (others=>LINE_ZERO);
        fifoIn(i) <= TEST_VECTOR(idx);
        wait until clkW'event and clkW = '1';
        if TEST_VECTOR(idx) /= LINE_ZERO then
          fifoIn(i) <= FLIT_STOP or std_logic_vector(to_unsigned(i, 35));
```

```
        else
          fifoIn(i) <= LINE_ZERO;
        end if;
        wait until clkW'event and clkW = '1';
      end loop;
    end loop;
    fifoIn <= (others=>LINE_ZERO);
    wait until reset = '1';
  end process wBehaviour;

  rBehaviour: process is
    variable outPort: integer;
  begin
    wait until reset = '1' and clkR'event and clkR = '1';
    wait until clkR'event and clkR = '1';
    wait until clkR'event and clkR = '1';
    wait until clkR'event and clkR = '1';
    wait until clkR'event and clkR = '1';

    -- one (and a half) period latency inherent in fifo
    wait until clkR'event and clkR = '1';
    -- two period latency due to pipeline in HPU
    wait until clkR'event and clkR = '1';
    wait until clkR'event and clkR = '1';

    for idx in 0 to TEST_LENGTH-1 loop
      report "Reading with outNum := " & str(OUT_NUM(idx)) severity note;
      for i in 0 to 4 loop
        -- check for correct output
        wait for 10 ns;
        if OUT_NUM(idx) = i then
          outPort := 4; -- local output
        else
          outPort := OUT_NUM(idx);
        end if;
        if routerOut(outPort) /= (TEST_VECTOR(idx)(34 downto 2) & "00") then
          report "Output mismatch header flit. idx := " & str(idx) & ", i := " &
              str(i) & ", outPort := " & str(outPort) severity error;
        end if;
        wait until clkR'event and clkR = '1';
        wait for 10 ns;
        if routerOut(outPort) /= (FLIT_STOP or std_logic_vector(to_unsigned(i,
            35))) and TEST_VECTOR(idx) /= LINE_ZERO then
          report "Output mismatch stop flit. idx := " & str(idx) & ", i := " &
              str(i) & ", outPort := " & str(outPort) severity error;
        end if;
        wait until clkR'event and clkR = '1';
      end loop;
    end loop;

    report "CONGRATULATIONS! If no failures, then all tests completed
        successfully!" severity note;
    wait until reset = '1';
  end process rBehaviour;

end behaviour;
```

vhdl/testPleso.vhd

```
-- testPleso.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Test bench for a plesiochronous system.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;
use work.txt_util.all;

entity testPleso is
end testPleso;

architecture behaviour of testPleso is
  signal clkW:              std_logic := '0';
  signal clkR:              std_logic := '0';
  signal reset:             std_logic;
  signal routerIn, routerOut:     XbarPort; -- 0 is SOUTH, 1 is WEST, 2 is NORTH,
      3 is EAST, 4 is LOCAL

  -- test vectors
  constant OUT_NORTH: dataLine := "11000000000000000000000000000000000";
  constant OUT_EAST: dataLine := "11000000000000000000000000000000001";
  constant OUT_SOUTH: dataLine := "11000000000000000000000000000000010";
  constant OUT_WEST: dataLine := "11000000000000000000000000000000011";
  constant FLIT_STOP: dataLine := "10100000000000000000000000000000000";

begin
  reset <= '0', '1' after 37 ns;

  clkW <= not clkW after 50 ns;

  Clk: process is
    variable count: integer := 0;
    variable skew: integer := 0;
  begin
    clkR <= not clkR;
    wait for 50 ns;
    if count = 9 then
      wait for 1 ns;
      count := 0;
      skew := skew + 1;
    else
      count := count + 1;
    end if;
  end process Clk;

  router:
  entity work.routerFifo
    port map(clkLocal=>clkR, clkNeighbour=>clkW, reset=>reset, inPort=>routerIn,
        outPort=>routerOut);

  wBehaviour: process is
    variable sequence: integer := 0;
  begin
    routerIn <= (others=>(others=>'0'));
    wait until reset = '1' and clkW'event and clkW = '1';

    loop
      routerIn(0) <= OUT_NORTH;
      wait until clkW'event and clkW = '1';
      routerIn(0) <= FLIT_STOP or std_logic_vector(to_unsigned(sequence, 35));
```

```
      sequence := sequence + 1;
      wait until clkW'event and clkW = '1';
    end loop;
  end process wBehaviour;

  rBehaviour: process is
    variable sequence: integer := 0;
  begin
    wait until reset = '1' and clkR'event and clkR = '1';

    -- one (and a half) period latency inherent in fifo
    wait until clkR'event and clkR = '1';
    -- two period latency due to pipeline in HPU
    wait until clkR'event and clkR = '1';
    wait until clkR'event and clkR = '1';

    loop
      wait for 3 ns;
      if routerOut(2) /= (OUT_NORTH(34 downto 2) & "00") then
        report "Received invalid header flit! Sequence is " & str(sequence) & "."
          severity failure;
      end if;
      wait until clkR'event and clkR = '1';
      wait for 3 ns;
      if routerOut(2) /= (FLIT_STOP or std_logic_vector(to_unsigned(sequence, 35)
          )) then
        report "Received invalid data flit! Sequence is " & str(sequence) & "."
          severity failure;
      end if;
      sequence := sequence + 1;
      wait until clkR'event and clkR = '1';
    end loop;
end process rBehaviour;

end behaviour;
```

vhdl/gatedRouterFifo.vhd

```
-- gatedRouterFifo.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Mesochronous router with clock gating.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity gatedRouterFifo is
  port ( clkLocal:     in  std_logic;
         clkNeighbour: in  std_logic;
         reset:       in  std_logic;
         inPort:      in  XbarPort;
         outPort:     out XbarPort
       );
end gatedRouterFifo;

architecture structure of gatedRouterFifo is
  signal fifoOut: XbarPort;
  signal fifoFull, fifoEmpty: std_logic_vector(4 downto 0);

begin
  fifoGen:
  for i in 0 to 4 generate
    gatedFifo:
    entity work.gatedFifo
      generic map(N=>5, W=>35)
      port map(clkW=>ClkNeighbour, clkR=>ClkLocal, reset=>reset, enable=>inPort(i
          )(34),
      dataW=>inPort(i), dataR=>fifoOut(i), full=>fifoFull(i), empty=>fifoEmpty(i)
          );
  end generate;

  gatedRouter:
  entity work.gatedRouter
    port map(clk=>clkLocal, reset=>reset, inPort=>fifoOut, outPort=>outPort);
end structure;
```

A.4 FPGA Implementation and Test

vhdl/fpgaTest.vhd

```
-- fpgaTest.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- FPGA test suite, proof-of-concept NoC router on FPGA.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.types.all;

entity TestEnv is
  port (
      clk:    in std_logic;
      reset:    in std_logic;
      btnOk:    in std_logic;
      sw:     in std_logic_vector(7 downto 0);  -- switches
      Led:    out std_logic_vector(7 downto 0); -- LEDs
      an:     out std_logic_vector(3 downto 0); -- Anodes
      seg:    out std_logic_vector(7 downto 0)  -- Cathodes
    );
end TestEnv;

architecture behaviour of TestEnv is
  constant NUM_PORTS: integer := 5;           -- number of i/o ports

  constant OUT_NORTH: dataLine := "11000000000000000000000000000000";
  constant OUT_EAST:  dataLine := "11000000000000000000000000000001";
  constant OUT_SOUTH: dataLine := "11000000000000000000000000000010";
  constant OUT_WEST:  dataLine := "11000000000000000000000000000011";
  constant FLIT_STOP: dataLine := "10100000000000000000000000000000";

  -- Test bench diagnostics
  -- Number of sent/received flits
  signal numSent, numSentNext: integer range 0 to 200;
  type recvType is array(0 to NUM_PORTS) of integer range 0 to 200;
  signal numRecvd, numRecvdNext: recvType;  -- received at each port + errors
  -- Control signals for FIFOs (data sent/recvd memory)
  type fifoInOutType is array (0 to NUM_PORTS-1) of dataLine;
  signal fifoIn, fifoOut: fifoInOutType;
  signal fifoFull, fifoEmpty, fifoWen, fifoRen: std_logic_vector(0 to NUM_PORTS
      -1);
  -- Output to seven segment display on Nexys2 board
  signal displayData: std_logic_vector(15 downto 0);

  -- Test bench state machines
  type sendStateType is (idle, sendStart, sendStop, sendDone);
  type recvStateType is (idle, recvStart, recvStop);
  type recvStateTypeArray is array(0 to NUM_PORTS-1) of recvStateType;

  signal sendState, sendStateNext:      sendStateType;
  signal sendDest, sendDestNext:        integer range 0 to NUM_PORTS-1;
  signal sendOrg, sendOrgNext:        integer range 0 to NUM_PORTS-1;
  signal serial, serialNext:          natural;
  signal sendHeader:              dataLine;
  signal recvState, recvStateNext:      recvStateTypeArray;
  signal recvBuf, recvBufNext:        XbarPort;
  signal routerIn, routerOut:         XbarPort;  -- 0 is SOUTH, 1 is WEST, 2 is
      NORTH, 3 is EAST, 4 is LOCAL

  signal clkW, clkR, clkDiv, clkBufG, clk0Out, clkLockedOut: std_logic;
  signal resetInv:  std_logic;
begin
  clkW <= clkDiv;
  clkR <= not clkDiv;

  resetInv <= not reset;
  --Led <= sw;
  Led <= (others => '0');

  -- decoder, numerical destination into header flit
  sendDestProc:
  process(sendDest)
  begin
    case sendDest is
      when 0 => sendHeader <= OUT_SOUTH;
      when 1 => sendHeader <= OUT_WEST;
      when 2 => sendHeader <= OUT_NORTH;
      when 3 => sendHeader <= OUT_EAST;
      when others => sendHeader <= (others => '0');
    end case;
  end process sendDestProc;

  sendProc:
  process(sendState, numSent, sendOrg, sendDest, sendHeader, serial)
    variable fifoDest: integer range 0 to NUM_PORTS-1;
  begin
    routerIn <= (others => (others => '0'));
    fifoIn <= (others => (others => '0'));
    fifoWen <= (others => '0');
    numSentNext <= numSent;
    sendOrgNext <= sendOrg;
    sendDestNext <= sendDest;
    serialNext <= serial;

    -- actual destination port
    if sendOrg = sendDest then
      fifoDest := 4;
    else
      fifoDest := sendDest;
    end if;

    case sendState is
      when idle =>
        sendStateNext <= sendStart;
      when sendStart =>
        routerIn(sendOrg) <= sendHeader;
        fifoIn(fifoDest) <= sendHeader(34 downto 2) & "00";
        fifoWen(fifoDest) <= '1';
        numSentNext <= numSent + 1;
        sendStateNext <= sendStop;
      when sendStop =>
        routerIn(sendOrg) <= (FLIT_STOP or std_logic_vector(to_unsigned(serial,
            35)));
```

```
      fifoIn(fifoDest) <= (FLIT_STOP or std_logic_vector(to_unsigned(serial,
          35)));
      serialNext <= serial + 1;
      fifoWen(fifoDest) <= '1';
      numSentNext <= numSent + 1;
      sendStateNext <= sendDone;
    when sendDone =>
      if sendOrg = 4 then
        if sendDest = 3 then
          sendStateNext <= sendDone;    -- done, remain in this state
        else
          sendDestNext <= sendDest + 1;
          sendOrgNext <= 0;
          sendStateNext <= sendStart;
        end if;
      else
        sendOrgNext <= sendOrg + 1;
        sendStateNext <= sendStart;
      end if;
  end case;
end process sendProc;

recvProc:
process(recvState, routerOut, numRecvd, fifoOut, recvBuf)
begin
  fifoRen <= (others => '0');
  numRecvdNext <= numRecvd;
  recvBufNext <= routerOut;

  -- generate state machines for every output port
  for i in recvState'range loop
    case recvState(i) is
      when idle =>
        if routerOut(i)(34) = '0' then
          recvStateNext(i) <= idle;
        else
          recvStateNext(i) <= recvStart;
        end if;
      when recvStart =>
        fifoRen(i) <= '1';
        recvStateNext(i) <= recvStop;
        if recvBuf(i) = fifoOut(i) then
          -- match
          numRecvdNext(i) <= numRecvd(i) + 1;
        else
          numRecvdNext(5) <= numRecvd(5) + 1;
        end if;
      when recvStop =>
        fifoRen(i) <= '1';
        recvStateNext(i) <= idle;
        if recvBuf(i) = fifoOut(i) then
          -- match
          numRecvdNext(i) <= numRecvd(i) + 1;
        else
          numRecvdNext(5) <= numRecvd(5) + 1; -- count number of errors
        end if;
    end case;
  end loop;
end process recvProc;

process(clkW, reset)
begin
  if reset = '1' then
    numSent <= 0;
    sendState <= idle;
    sendOrg <= 0;
    sendDest <= 0;
    serial <= 1024;
  elsif clkW'event and clkW = '1' then
    numSent <= numSentNext;
    sendState <= sendStateNext;
    sendOrg <= sendOrgNext;
    sendDest <= sendDestNext;
    serial <= serialNext;
  end if;
end process;

process(clkR, reset)
begin
  if reset = '1' then
    numRecvd <= (others => 0);
    numRecvd(5) <= 16;  -- just to see something in display
    for i in recvState'range loop
      recvState(i) <= idle;
    end loop;
    --recvState <= (others => idle); generates spurious width mismatch warnings
    recvBuf <= (others => (others => '0'));
  elsif clkR'event and clkR = '1' then
    numRecvd <= numRecvdNext;
    recvState <= recvStateNext;
    recvBuf <= recvBufNext;
  end if;
end process;

process(sw, numSent, numRecvd)
begin
  displayData <= std_logic_vector(to_unsigned(numSent, 8)) & "00000000";
  -- inverse priority decoder
  -- depending on switches, show # of received flits at each port
  -- numRecvd(5) is # of errors
  for i in numRecvd'range loop
    if sw(i) = '1' then
      displayData(7 downto 0) <= std_logic_vector(to_unsigned(numRecvd(i), 8));
    end if;
  end loop;
end process;

-- 7-segment display
display:
entity work.sevseg
  port map(clk=>clkW, rst=>reset, val=>displayData, seg0=>"0000", seg1=>"0000",
       seg2=>"0000", seg3=>"0000",
  dp=>"0000", wen=>'1', wendp=>"0000", wenseg=>"0000", useseg=>'0', anout=>an,
       ctout=>seg);

-- DUT (router)
router:
entity work.routerFifo
  port map(clkLocal=>clkR, clkNeighbour=>clkW, reset=>resetInv, inPort=>
       routerIn, outPort=>routerOut);

-- FIFOs to keep track of what has been sent
fifoGen:
for i in 0 to NUM_PORTS-1 generate
  fifo:entity work.FIFO
    generic map(N=>10, w=>35)
    port map(clkW=>clkW, clkR=>clkR, reset=>resetInv, writeEn=>fifoWen(i),
         readEn=>fifoRen(i),
```

```
      dataW=>fifoIn(i), dataR=>fifoOut(i), full=>fifoFull(i), empty=>fifoEmpty(i)
          );
  end generate;

  -- DCM (clock divider). Xilinx IP
  DCM:
  entity work.ClockDivider
    port map (clkIn_In=>clk, rst_in=>reset, clkdv_Out=>clkDiv, clkin_Ibufg_out=>
        clkBufG, clk0_out=>clk0out, locked_out=>clkLockedOut);
end behaviour;
```

vhdl/fpgaTestSim.vhd

```
-- fpgaTestSim.vhd
-- A. Bentzon, 2012. BSc thesis, 'Mesochronous TDM-based Network-on-Chip'.

-- Wrapper used to simulate the FPGA test environment.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity TestEnvSim is
end TestEnvSim;

architecture structure of TestEnvSim is
  signal clk:     std_logic := '0';
  signal reset:   std_logic;

  -- dummy signals
  signal Led, seg:  std_logic_vector(7 downto 0);
  signal an:      std_logic_vector(3 downto 0);
begin
  reset <= '1', '0' after 537 ns;
  clk <= not clk after 20 ns;

  testEnv:
  entity work.TestEnv
    port map(clk=>clk, reset=>reset, btnOk=>'0', sw=>(others=>'0'), Led=>Led, seg
        =>seg);

end structure;
```

Appendix B

Redacted synthesis reports

To save space, the reports generated by XST have been redacted to show only the relevant information. The full reports are available upon request.

B.1 The Synchronous Network

synth/router.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing Entity <router> in library <work> (Architecture <struct>).
Entity <router> analyzed. Unit <router> generated.

Analyzing Entity <HPU> in library <work> (Architecture <struct>).
Entity <HPU> analyzed. Unit <HPU> generated.

Analyzing Entity <Xbar> in library <work> (Architecture <structure>).
Entity <Xbar> analyzed. Unit <Xbar> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...
```

```
Synthesizing Unit <HPU>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        hpu.vhd".
    Found 4-bit register for signal <selInt>.
    Summary:
  inferred   4 D-type flip-flop(s).
Unit <HPU> synthesized.


Synthesizing Unit <Xbar>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        xbar.vhd".
Unit <Xbar> synthesized.


Synthesizing Unit <router>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        router.vhd".
    Found 175-bit register for signal <HPUout>.
    Found 175-bit register for signal <XbarOut>.
    Found 20-bit register for signal <XbarSel>.
    Summary:
  inferred 370 D-type flip-flop(s).
Unit <router> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 16
 20-bit register                                     : 1
 35-bit register                                     : 10
 4-bit register                                      : 5

=========================================================================


=========================================================================
*                            Advanced HDL Synthesis                     *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 390
 Flip-Flops                                          : 390

=========================================================================

[...]

=========================================================================
*                                Final Report                           *
=========================================================================
Final Results
RTL Top Level Output File Name     : router.ngr
Top Level Output File Name         : router
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 352

Cell Usage :
# BELS                             : 761
#      INV                         : 1
#      LUT2                        : 220
#      LUT3                        : 150
#      LUT4                        : 215
#      LUT4_L                      : 175
# FlipFlops/Latches                : 410
#      FDC                         : 410
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 351
#      IBUF                        : 176
#      OBUF                        : 175
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                      414  out of   8672     4%
 Number of Slice Flip Flops:            410  out of  17344     2%
 Number of 4 input LUTs:                761  out of  17344     4%
 Number of IOs:                         352
 Number of bonded IOBs:                 352  out of    250   140% (*)
 Number of GCLKs:                         1  out of     24     4%

[...]

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 3.909ns (Maximum Frequency: 255.820MHz)
   Minimum input arrival time before clock: 5.136ns
   Maximum output required time after clock: 4.283ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 3.909ns (frequency: 255.820MHz)
  Total number of paths / destination ports: 1460 / 235
-------------------------------------------------------------------------
Delay:               3.909ns (Levels of Logic = 2)
  Source:            XbarSel_10_1 (FF)
  Destination:       XbarOut_0_2 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: XbarSel_10_1 to XbarOut_0_2
                                Gate     Net
    Cell:in->out      fanout   Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q             18    0.591    1.103  XbarSel_10_1 (XbarSel_10_1)
     LUT4:I2->O            1    0.704    0.499  xbar/outPort_0_or0000<9>9 (xbar/
```

```
    outPort_0_or0000<9>9)
  LUT2:I1->O            1    0.704    0.000   xbar/outPort_0_or0000<9>10 (
    XbarOutNext<0><9>)
  FDC:D                      0.308            XbarOut_0_9
  ----------------------------------------
```

```
    Total                          3.909ns (2.307ns logic, 1.602ns route)
                                           (59.0% logic, 41.0% route)

[...]
```

synth/HPU.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing Entity <HPU> in library <work> (Architecture <struct>).
Entity <HPU> analyzed. Unit <HPU> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <HPU>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        hpu.vhd".
    Found 4-bit register for signal <selInt>.
    Summary:
  inferred    4 D-type flip-flop(s).
Unit <HPU> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 1
 4-bit register                                      : 1

=========================================================================


=========================================================================
*                         Advanced HDL Synthesis                        *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
```

```
# Registers                                          : 4
 Flip-Flops                                          : 4

=========================================================================

[...]

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : HPU.ngr
Top Level Output File Name         : HPU
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 76

Cell Usage :
# BELS                             : 48
#      INV                         : 1
#      LUT2                        : 9
#      LUT3                        : 30
#      LUT4                        : 8
# FlipFlops/Latches                : 4
#      FDC                         : 4
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 75
#      IBUF                        : 36
#      OBUF                        : 39
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                        27  out of   8672     0%
 Number of Slice Flip Flops:               4  out of  17344     0%
 Number of 4 input LUTs:                  48  out of  17344     0%
 Number of IOs:                           76
 Number of bonded IOBs:                   76  out of    250    30%
 Number of GCLKs:                          1  out of     24     4%

[...]
```

synth/Xbar.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================

Analyzing Entity <Xbar> in library <work> (Architecture <structure>).
Entity <Xbar> analyzed. Unit <Xbar> generated.

[...]

=========================================================================
*                            Final Report                               *
=========================================================================

Final Results
RTL Top Level Output File Name     : Xbar.ngr
Top Level Output File Name         : Xbar
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 370

Cell Usage :
# BELS                             : 525
#      LUT2                        : 175
#      LUT4                        : 350
# IO Buffers                       : 370
#      IBUF                        : 195
#      OBUF                        : 175
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                      302  out of   8672     3%
 Number of 4 input LUTs:                525  out of  17344     3%
 Number of IOs:                         370
 Number of bonded IOBs:                 370  out of    250   148% (*)

[...]
```

synth/gatedRouter.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================

Analyzing Entity <gatedRouter> in library <work> (Architecture <structure>).
Entity <gatedRouter> analyzed. Unit <gatedRouter> generated.

Analyzing Entity <router> in library <work> (Architecture <struct>).
Entity <router> analyzed. Unit <router> generated.

Analyzing Entity <HPU> in library <work> (Architecture <struct>).
Entity <HPU> analyzed. Unit <HPU> generated.

Analyzing Entity <Xbar> in library <work> (Architecture <structure>).
Entity <Xbar> analyzed. Unit <Xbar> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <HPU>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        hpu.vhd".
    Found 4-bit register for signal <selInt>.
    Summary:
  inferred   4 D-type flip-flop(s).
Unit <HPU> synthesized.


Synthesizing Unit <Xbar>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        xbar.vhd".
Unit <Xbar> synthesized.


Synthesizing Unit <router>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        router.vhd".
    Found 175-bit register for signal <HPUout>.
    Found 175-bit register for signal <XbarOut>.
    Found 20-bit register for signal <XbarSel>.
    Summary:
  inferred 370 D-type flip-flop(s).
Unit <router> synthesized.


Synthesizing Unit <gatedRouter>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        gatedRouter.vhd".
WARNING:Xst:737 - Found 1-bit latch for signal <clkEn>. Latches may be generated
     from incomplete case or if statements. We do not recommend the use of
     latches in FPGA/CPLD designs, as they may lead to timing problems.
    Found 1-bit register for signal <validSigOut1>.
    Found 1-bit register for signal <validSigOut2>.
    Summary:
  inferred   2 D-type flip-flop(s).
Unit <gatedRouter> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
```

```
# Registers                                          : 18
 1-bit register                                      : 2
 20-bit register                                     : 1
 35-bit register                                     : 10
 4-bit register                                      : 5
# Latches                                            : 1
 1-bit latch                                         : 1

=========================================================================

=========================================================================
*                            Advanced HDL Synthesis                     *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
     :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 392
 Flip-Flops                                          : 392
# Latches                                            : 1
 1-bit latch                                         : 1

=========================================================================

[...]

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : gatedRouter.ngr
Top Level Output File Name         : gatedRouter
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 352

Cell Usage :
# BELS                             : 766
#      INV                         : 1
#      LUT2                        : 222
#      LUT3                        : 150
#      LUT4                        : 216
#      LUT4_L                      : 175
#      MUXF5                       : 1
#      VCC                         : 1
# FlipFlops/Latches                : 413
#      FDC                         : 410
#      FDP                         : 2
#      LD_1                        : 1
# Clock Buffers                    : 2
#      BUFG                        : 2
# IO Buffers                       : 352
#      IBUF                        : 177
#      OBUF                        : 175
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                        416  out of   8672     4%
 Number of Slice Flip Flops:              413  out of  17344     2%
 Number of 4 input LUTs:                  764  out of  17344     4%
 Number of IOs:                           352
 Number of bonded IOBs:                   352  out of    250   140% (*)
 Number of GCLKs:                           2  out of     24     8%

[...]

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 3.909ns (Maximum Frequency: 255.820MHz)
   Minimum input arrival time before clock: 5.136ns
   Maximum output required time after clock: 4.283ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 2.102ns (frequency: 475.737MHz)
  Total number of paths / destination ports: 2 / 2
-------------------------------------------------------------------------
Delay:               2.102ns (Levels of Logic = 1)
  Source:            validSigOut2 (FF)
  Destination:       clkEn (LATCH)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: validSigOut2 to clkEn
                                   Gate     Net
    Cell:in->out      fanout     Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDP:C->Q             1     0.591    0.499  validSigOut2 (validSigOut2)
     LUT2:I1->O           1     0.704    0.000  gateEnable1 (gateEnable)
     LD_1:D                     0.308           clkEn
    ----------------------------------------
    Total                      2.102ns (1.603ns logic, 0.499ns route)
                                       (76.3% logic, 23.7% route)

=========================================================================
Timing constraint: Default period analysis for Clock 'gatedClk1'
  Clock period: 3.909ns (frequency: 255.820MHz)
  Total number of paths / destination ports: 1460 / 235
-------------------------------------------------------------------------
Delay:               3.909ns (Levels of Logic = 2)
  Source:            router/XbarSel_7_1 (FF)
  Destination:       router/XbarOut_4_34 (FF)
  Source Clock:      gatedClk1 rising
  Destination Clock: gatedClk1 rising

  Data Path: router/XbarSel_7_1 to router/XbarOut_4_34
                                   Gate     Net
    Cell:in->out      fanout     Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
```

```
    FDC:C->Q              18    0.591   1.103   router/XbarSel_7_1 (router/
        XbarSel_7_1)
    LUT4:I2->O             1    0.704   0.499   router/xbar/outPort_4_or0000<9>9 (
        router/xbar/outPort_4_or0000<9>9)
    LUT2:I1->O             1    0.704   0.000   router/xbar/outPort_4_or0000<9>10 (
        router/XbarOutNext<4><9>)
    FDC:D                       0.308          router/XbarOut_4_9
    ----------------------------------------
    Total                       3.909ns (2.307ns logic, 1.602ns route)
                                        (59.0% logic, 41.0% route)

[...]
```

B.2 A FIFO Synchroniser for Mesochronous Networks

synth/fifo.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing generic Entity <fifo> in library <work> (Architecture <structure>).
  N = 5
  W = 35
Entity <fifo> analyzed. Unit <fifo> generated.

Analyzing generic Entity <tokenRing.1> in library <work> (Architecture <behaviour
    >).
  N = 5
  default = 3
Entity <tokenRing.1> analyzed. Unit <tokenRing.1> generated.

Analyzing generic Entity <tokenRing.2> in library <work> (Architecture <behaviour
    >).
  N = 5
  default = 12
Entity <tokenRing.2> analyzed. Unit <tokenRing.2> generated.

Analyzing generic Entity <fullDetector> in library <work> (Architecture <
    behaviour>).
  N = 5
Entity <fullDetector> analyzed. Unit <fullDetector> generated.

Analyzing generic Entity <emptyDetector> in library <work> (Architecture <
    structure>).
  N = 5
Entity <emptyDetector> analyzed. Unit <emptyDetector> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <tokenRing_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_1> synthesized.


Synthesizing Unit <tokenRing_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_2> synthesized.

Synthesizing Unit <fullDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fulldetector.vhd".
    Found 1-bit register for signal <sync0>.
    Found 1-bit register for signal <sync1>.
    Found 1-bit register for signal <sync2>.
    Summary:
  inferred   3 D-type flip-flop(s).
Unit <fullDetector> synthesized.


Synthesizing Unit <emptyDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        emptydetector.vhd".
    Found 5-bit register for signal <syncWriteP>.
    Summary:
  inferred   5 D-type flip-flop(s).
Unit <emptyDetector> synthesized.


Synthesizing Unit <fifo>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fifo.vhd".
    Found 175-bit register for signal <dataBuf>.
    Summary:
  inferred 175 D-type flip-flop(s).
Unit <fifo> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 11
 1-bit register                                      : 3
 35-bit register                                     : 5
 5-bit register                                      : 3
=========================================================================

=========================================================================
*                        Advanced HDL Synthesis                         *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 193
 Flip-Flops                                          : 193
=========================================================================

[...]
```

```
*                          Final Report                          *

Final Results
RTL Top Level Output File Name     : fifo.ngr
Top Level Output File Name         : fifo
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 77

Cell Usage :
# BELS                             : 265
#      GND                         : 1
#      INV                         : 1
#      LUT2                        : 17
#      LUT2_L                      : 1
#      LUT3                        : 57
#      LUT4                        : 135
#      LUT4_D                      : 1
#      LUT4_L                      : 1
#      MUXF5                       : 51
# FlipFlops/Latches                : 196
#      FDC                         : 8
#      FDCE                        : 183
#      FDPE                        : 5
# Clock Buffers                    : 2
#      BUFGP                       : 2
# IO Buffers                       : 75
#      IBUF                        : 38
#      OBUF                        : 37

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                      167  out of   8672     1%
 Number of Slice Flip Flops:            196  out of  17344     1%
 Number of 4 input LUTs:                213  out of  17344     1%
 Number of IOs:                          77
 Number of bonded IOBs:                  77  out of    250    30%
 Number of GCLKs:                         2  out of     24     8%

[...]

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 5.300ns (Maximum Frequency: 188.674MHz)
   Minimum input arrival time before clock: 5.983ns
   Maximum output required time after clock: 10.166ns
   Maximum combinational path delay: 8.378ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clkW'
  Clock period: 5.300ns (frequency: 188.674MHz)
  Total number of paths / destination ports: 722 / 188
-------------------------------------------------------------------------
Delay:               5.300ns (Levels of Logic = 2)
  Source:            fullDet/sync1 (FF)
  Destination:       dataBuf_0_0 (FF)
  Source Clock:      clkW rising
  Destination Clock: clkW rising

  Data Path: fullDet/sync1 to dataBuf_0_0
                                   Gate     Net
    Cell:in->out      fanout       Delay    Delay   Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q             3       0.591    0.566   fullDet/sync1 (fullDet/sync1)
     LUT3:I2->O          10       0.704    0.917   writeEnInt1 (writeEnInt)
     LUT3:I2->O          35       0.704    1.263   dataBuf_4_and00001 (
         dataBuf_4_and0000)
     FDCE:CE                      0.555            dataBuf_4_0
    ----------------------------------------
    Total                         5.300ns (2.554ns logic, 2.746ns route)
                                          (48.2% logic, 51.8% route)

=========================================================================
Timing constraint: Default period analysis for Clock 'clkR'
  Clock period: 5.223ns (frequency: 191.461MHz)
  Total number of paths / destination ports: 144 / 16
-------------------------------------------------------------------------
Delay:               5.223ns (Levels of Logic = 3)
  Source:            readP/ring_1_1 (FF)
  Destination:       readP/ring_2 (FF)
  Source Clock:      clkR rising
  Destination Clock: clkR rising

  Data Path: readP/ring_1_1 to readP/ring_2
                                   Gate     Net
    Cell:in->out      fanout       Delay    Delay   Logical Name (Net Name)
    ----------------------------------------  ------------
     FDCE:C->Q            2       0.591    0.622   readP/ring_1_1 (readP/ring_1_1)
     LUT2_L:I0->LO        1       0.704    0.104   emptyDet/empty71_SW0_SW0 (N48)
     LUT4:I3->O           2       0.704    0.482   emptyDet/empty71_SW0 (N46)
     LUT4:I2->O           8       0.704    0.757   dataR<0>310_1 (dataR<0>310)
     FDPE:CE                      0.555            readP/ring_2
    ----------------------------------------
    Total                         5.223ns (3.258ns logic, 1.965ns route)
                                          (62.4% logic, 37.6% route)

[...]
```

synth/tokenring.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing generic Entity <tokenRing> in library <work> (Architecture <behaviour>)
    .
    N = 5
    default = 1
Entity <tokenRing> analyzed. Unit <tokenRing> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <tokenRing>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 1
 5-bit register                                      : 1

=========================================================================


=========================================================================
*                        Advanced HDL Synthesis                         *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 5
 Flip-Flops                                          : 5

=========================================================================

[...]

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : tokenRing.ngr
Top Level Output File Name         : tokenRing
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 8

Cell Usage :
# BELS                             : 1
#      INV                         : 1
# FlipFlops/Latches                : 5
#      FDCE                        : 4
#      FDPE                        : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 7
#      IBUF                        : 2
#      OBUF                        : 5
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                        3  out of   8672     0%
 Number of Slice Flip Flops:              5  out of  17344     0%
 Number of 4 input LUTs:                  1  out of  17344     0%
 Number of IOs:                           8
 Number of bonded IOBs:                   8  out of    250     3%
 Number of GCLKs:                         1  out of     24     4%

[...]
```

synth/fullDetector.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing generic Entity <fullDetector> in library <work> (Architecture <
    behaviour>).
    N = 5
Entity <fullDetector> analyzed. Unit <fullDetector> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <fullDetector>.
```

```
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fulldetector.vhd".
    Found 1-bit register for signal <sync0>.
    Found 1-bit register for signal <sync1>.
    Found 1-bit register for signal <sync2>.
    Summary:
  inferred    3 D-type flip-flop(s).
Unit <fullDetector> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 3
 1-bit register                                      : 3

=========================================================================


=========================================================================
*                            Advanced HDL Synthesis                     *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 3
 Flip-Flops                                          : 3

=========================================================================

[...]

=========================================================================
*                                Final Report                           *
=========================================================================
Final Results
RTL Top Level Output File Name     : fullDetector.ngr
Top Level Output File Name         : fullDetector
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 14

Cell Usage :
# BELS                             : 6
#      INV                         : 1
#      LUT2                        : 2
#      LUT4                        : 3
# FlipFlops/Latches                : 3
#      FDC                         : 3
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 13
#      IBUF                        : 12
#      OBUF                        : 1
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                        3  out of   8672     0%
 Number of Slice Flip Flops:              3  out of  17344     0%
 Number of 4 input LUTs:                  6  out of  17344     0%
 Number of IOs:                          14
 Number of bonded IOBs:                  14  out of    250     5%
 Number of GCLKs:                         1  out of     24     4%

[...]
```

synth/emptyDetector.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing generic Entity <emptyDetector> in library <work> (Architecture <
    structure>).
  N = 5
Entity <emptyDetector> analyzed. Unit <emptyDetector> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <emptyDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        emptydetector.vhd".
    Found 5-bit register for signal <syncWriteP>.
    Summary:
  inferred    5 D-type flip-flop(s).
Unit <emptyDetector> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 1
 5-bit register                                      : 1

=========================================================================


=========================================================================
*                            Advanced HDL Synthesis                     *
=========================================================================
```

```
Loading device for application Rf_Device from file '3s1200e.nph' in environment C
     :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 5
 Flip-Flops                                         : 5

=========================================================================

[...]

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : emptyDetector.ngr
Top Level Output File Name         : emptyDetector
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 13

Cell Usage :
# BELS                             : 10
#      INV                         : 1
#      LUT2                        : 2
#      LUT3                        : 3
#      LUT4                        : 2
#      MUXF5                       : 2
# FlipFlops/Latches                : 5
#      FDC                         : 5
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 12
#      IBUF                        : 11
#      OBUF                        : 1
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                        4  out of   8672     0%
 Number of Slice Flip Flops:              5  out of  17344     0%
 Number of 4 input LUTs:                  8  out of  17344     0%
 Number of IOs:                          13
 Number of bonded IOBs:                  13  out of    250     5%
    IOB Flip Flops:                       5
 Number of GCLKs:                         1  out of     24     4%

[...]
```

synth/gatedFifo.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing generic Entity <gatedFifo> in library <work> (Architecture <structure>)
    .
    N = 5
    W = 35
Entity <gatedFifo> analyzed. Unit <gatedFifo> generated.

Analyzing generic Entity <fifo> in library <work> (Architecture <structure>).
    N = 5
    W = 35
Entity <fifo> analyzed. Unit <fifo> generated.

Analyzing generic Entity <tokenRing.1> in library <work> (Architecture <behaviour
    >).
    N = 5
    default = 3
Entity <tokenRing.1> analyzed. Unit <tokenRing.1> generated.

Analyzing generic Entity <tokenRing.2> in library <work> (Architecture <behaviour
    >).
    N = 5
    default = 12
Entity <tokenRing.2> analyzed. Unit <tokenRing.2> generated.

Analyzing generic Entity <fullDetector> in library <work> (Architecture <
    behaviour>).
    N = 5
Entity <fullDetector> analyzed. Unit <fullDetector> generated.

Analyzing generic Entity <emptyDetector> in library <work> (Architecture <
    structure>).
    N = 5
Entity <emptyDetector> analyzed. Unit <emptyDetector> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <tokenRing_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_1> synthesized.


Synthesizing Unit <tokenRing_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_2> synthesized.
```

```
Synthesizing Unit <fullDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fulldetector.vhd".
    Found 1-bit register for signal <sync0>.
    Found 1-bit register for signal <sync1>.
    Found 1-bit register for signal <sync2>.
    Summary:
  inferred   3 D-type flip-flop(s).
Unit <fullDetector> synthesized.


Synthesizing Unit <emptyDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        emptydetector.vhd".
    Found 5-bit register for signal <syncWriteP>.
    Summary:
  inferred   5 D-type flip-flop(s).
Unit <emptyDetector> synthesized.


Synthesizing Unit <fifo>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fifo.vhd".
    Found 175-bit register for signal <dataBuf>.
    Summary:
  inferred 175 D-type flip-flop(s).
Unit <fifo> synthesized.


Synthesizing Unit <gatedFifo>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        gatedFifo.vhd".
WARNING:Xst:737 - Found 1-bit latch for signal <clkWEn>. Latches may be generated
      from incomplete case or if statements. We do not recommend the use of
     latches in FPGA/CPLD designs, as they may lead to timing problems.
Unit <gatedFifo> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 11
 1-bit register                                      : 3
 35-bit register                                     : 5
 5-bit register                                      : 3
# Latches                                            : 1
 1-bit latch                                         : 1

=========================================================================


=========================================================================
*                            Advanced HDL Synthesis                       *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 193
 Flip-Flops                                          : 193
# Latches                                            : 1
 1-bit latch                                         : 1

=========================================================================

[...]

=========================================================================
*                            Final Report                                 *
=========================================================================
Final Results
RTL Top Level Output File Name     : gatedFifo.ngr
Top Level Output File Name         : gatedFifo
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 76

Cell Usage :
# BELS                             : 151
#      GND                         : 1
#      INV                         : 1
#      LUT2                        : 3
#      LUT3                        : 23
#      LUT4                        : 119
#      LUT4_D                      : 1
#      LUT4_L                      : 1
#      MUXF5                       : 1
#      VCC                         : 1
# FlipFlops/Latches                : 195
#      FDC                         : 8
#      FDCE                        : 182
#      FDPE                        : 4
#      LD_1                        : 1
# Clock Buffers                    : 3
#      BUFG                        : 2
#      BUFGP                       : 1
# IO Buffers                       : 75
#      IBUF                        : 38
#      OBUF                        : 37
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                      115  out of   8672     1%
 Number of Slice Flip Flops:            194  out of  17344     1%
 Number of 4 input LUTs:                148  out of  17344     0%
 Number of IOs:                          76
 Number of bonded IOBs:                  76  out of    250    30%
    IOB Flip Flops:                       1
 Number of GCLKs:                         3  out of     24    12%

[...]

Timing Summary:
---------------
Speed Grade: -4
```

```
   Minimum period: 5.190ns (Maximum Frequency: 192.678MHz)
   Minimum input arrival time before clock: 2.159ns
   Maximum output required time after clock: 14.904ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clkR'
  Clock period: 5.190ns (frequency: 192.678MHz)
  Total number of paths / destination ports: 114 / 12
-------------------------------------------------------------------------
Delay:               5.190ns (Levels of Logic = 3)
  Source:            fifo/readP/ring_3 (FF)
  Destination:       fifo/readP/ring_4 (FF)
  Source Clock:      clkR rising
  Destination Clock: clkR rising

  Data Path: fifo/readP/ring_3 to fifo/readP/ring_4
                                   Gate     Net
    Cell:in->out      fanout     Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDPE:C->Q             6    0.591    0.704  fifo/readP/ring_3 (fifo/readP/
          ring_3)
     LUT3:I2->O            1    0.704    0.455  fifo/emptyDet/empty9 (fifo/emptyDet
          /empty9)
     LUT4_D:I2->LO         1    0.704    0.104  fifo/emptyDet/empty35 (N115)
     LUT4:I3->O            6    0.704    0.669  fifo/readEnInt1 (fifo/readEnInt)
     FDPE:CE                    0.555           fifo/readP/ring_2
    ----------------------------------------
    Total                      5.190ns (3.258ns logic, 1.932ns route)
                                       (62.8% logic, 37.2% route)

=========================================================================
Timing constraint: Default period analysis for Clock 'gatedClkW1'
  Clock period: 3.921ns (frequency: 255.027MHz)
  Total number of paths / destination ports: 721 / 187
-------------------------------------------------------------------------
Delay:               3.921ns (Levels of Logic = 1)
  Source:            fifo/writeP/ring_0 (FF)
  Destination:       fifo/dataBuf_4_34 (FF)
  Source Clock:      gatedClkW1 rising
  Destination Clock: gatedClkW1 rising

  Data Path: fifo/writeP/ring_0 to fifo/dataBuf_4_34
                                   Gate     Net
    Cell:in->out      fanout     Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDPE:C->Q             5    0.591    0.808  fifo/writeP/ring_0 (fifo/writeP/
          ring_0)
     LUT4:I0->O           35    0.704    1.263  fifo/dataBuf_4_and00001 (fifo/
          dataBuf_4_and0000)
     FDCE:CE                    0.555           fifo/dataBuf_4_0
    ----------------------------------------
    Total                      3.921ns (1.850ns logic, 2.071ns route)
                                       (47.2% logic, 52.8% route)

[...]
```

B.3 The Mesochronous Network

synth/routerFifo.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing Entity <routerFifo> in library <work> (Architecture <structure>).
Entity <routerFifo> analyzed. Unit <routerFifo> generated.

Analyzing generic Entity <fifo> in library <work> (Architecture <structure>).
    N = 5
    W = 35
Entity <fifo> analyzed. Unit <fifo> generated.

Analyzing generic Entity <tokenRing.1> in library <work> (Architecture <behaviour
    >).
    N = 5
    default = 3
Entity <tokenRing.1> analyzed. Unit <tokenRing.1> generated.

Analyzing generic Entity <tokenRing.2> in library <work> (Architecture <behaviour
    >).
    N = 5
    default = 12
Entity <tokenRing.2> analyzed. Unit <tokenRing.2> generated.

Analyzing generic Entity <fullDetector> in library <work> (Architecture <
    behaviour>).
    N = 5
Entity <fullDetector> analyzed. Unit <fullDetector> generated.

Analyzing generic Entity <emptyDetector> in library <work> (Architecture <
    structure>).
    N = 5
Entity <emptyDetector> analyzed. Unit <emptyDetector> generated.

Analyzing Entity <router> in library <work> (Architecture <struct>).
Entity <router> analyzed. Unit <router> generated.

Analyzing Entity <HPU> in library <work> (Architecture <struct>).
Entity <HPU> analyzed. Unit <HPU> generated.

Analyzing Entity <Xbar> in library <work> (Architecture <structure>).
Entity <Xbar> analyzed. Unit <Xbar> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <tokenRing_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_1> synthesized.


Synthesizing Unit <tokenRing_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_2> synthesized.


Synthesizing Unit <fullDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fulldetector.vhd".
    Found 1-bit register for signal <sync0>.
    Found 1-bit register for signal <sync1>.
    Found 1-bit register for signal <sync2>.
    Summary:
  inferred   3 D-type flip-flop(s).
Unit <fullDetector> synthesized.


Synthesizing Unit <emptyDetector>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        emptydetector.vhd".
    Found 5-bit register for signal <syncWriteP>.
    Summary:
  inferred   5 D-type flip-flop(s).
Unit <emptyDetector> synthesized.


Synthesizing Unit <HPU>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        hpu.vhd".
    Found 4-bit register for signal <selInt>.
    Summary:
  inferred   4 D-type flip-flop(s).
Unit <HPU> synthesized.


Synthesizing Unit <Xbar>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        xbar.vhd".
Unit <Xbar> synthesized.


Synthesizing Unit <fifo>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fifo.vhd".
    Found 175-bit register for signal <dataBuf>.
    Summary:
  inferred 175 D-type flip-flop(s).
Unit <fifo> synthesized.


Synthesizing Unit <router>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
```

```
        router.vhd".
    Found 175-bit register for signal <HPUout>.
    Found 175-bit register for signal <XbarOut>.
    Found 20-bit register for signal <XbarSel>.
    Summary:
  inferred 370 D-type flip-flop(s).
Unit <router> synthesized.


Synthesizing Unit <routerFifo>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        routerFifo.vhd".
WARNING:Xst:646 - Signal <fifoFull> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <fifoEmpty> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
Unit <routerFifo> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 71
 1-bit register                                      : 15
 20-bit register                                     : 1
 35-bit register                                     : 35
 4-bit register                                      : 5
 5-bit register                                      : 15

=========================================================================


=========================================================================
*                            Advanced HDL Synthesis                       *
=========================================================================

Loading device for application Rf_Device from file '3s1200e.nph' in environment C
    :\Program Files (x86)\Xilinx\ISE.

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 1355
 Flip-Flops                                          : 1355

=========================================================================

[...]

=========================================================================
*                            Final Report                                 *
=========================================================================
Final Results
RTL Top Level Output File Name     : routerFifo.ngr
Top Level Output File Name         : routerFifo
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 353

Cell Usage :
# BELS                             : 2262
#      INV                         : 3
#      LUT2                        : 184
#      LUT2_D                      : 11
#      LUT2_L                      : 15
#      LUT3                        : 385
#      LUT3_L                      : 25
#      LUT4                        : 1013
#      LUT4_D                      : 92
#      LUT4_L                      : 266
#      MUXF5                       : 268
# FlipFlops/Latches                : 1390
#      FDC                         : 430
#      FDCE                        : 925
#      FDPE                        : 35
# Clock Buffers                    : 2
#      BUFGP                       : 2
# IO Buffers                       : 351
#      IBUF                        : 176
#      OBUF                        : 175
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                     1450  out of   8672    16%
 Number of Slice Flip Flops:           1390  out of  17344     8%
 Number of 4 input LUTs:               1994  out of  17344    11%
 Number of IOs:                         353
 Number of bonded IOBs:                 353  out of    250   141% (*)
 Number of GCLKs:                         2  out of     24     8%

[...]

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 7.566ns (Maximum Frequency: 132.163MHz)
   Minimum input arrival time before clock: 7.695ns
   Maximum output required time after clock: 4.283ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clkLocal'
  Clock period: 7.566ns (frequency: 132.163MHz)
  Total number of paths / destination ports: 18238 / 510
-------------------------------------------------------------------------
Delay:               7.566ns (Levels of Logic = 5)
  Source:            fifoGen[1].fifo/readP/ring_2 (FF)
  Destination:       router/port1/selInt_0 (FF)
  Source Clock:      clkLocal rising
  Destination Clock: clkLocal rising

  Data Path: fifoGen[1].fifo/readP/ring_2 to router/port1/selInt_0
                                Gate     Net
    Cell:in->out      fanout   Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
```

```
    FDPE:C->Q             77    0.591    1.451   fifoGen[1].fifo/readP/ring_2 (
        fifoGen[1].fifo/readP/ring_2)
    LUT3:I0->O             1    0.704    0.000   fifoGen[1].fifo/dataR<0>365_F (N492
        )
    MUXF5:I0->O            3    0.321    0.535   fifoGen[1].fifo/dataR<0>365 (
        fifoGen[1].fifo/dataR<0>365)
    LUT4:I3->O            19    0.704    1.089   fifoGen[1].fifo/dataR<0>3113_2 (
        fifoGen[1].fifo/dataR<0>3113_1)
    LUT4:I3->O             1    0.704    0.455   router/port1/selIntNext<0>_SW1 (
        N401)
    LUT4:I2->O             1    0.704    0.000   router/port1/selIntNext<0> (router/
        port1/selIntNext<0>)
    FDC:D                       0.308            router/port1/selInt_0
    ----------------------------------------
    Total                       7.566ns (4.036ns logic, 3.530ns route)
                                        (53.3% logic, 46.7% route)

=========================================================================
Timing constraint: Default period analysis for Clock 'clkNeighbour'
  Clock period: 5.260ns (frequency: 190.108MHz)
  Total number of paths / destination ports: 3610 / 940
-------------------------------------------------------------------------
```

```
Delay:                  5.260ns (Levels of Logic = 2)
  Source:               fifoGen[4].fifo/fullDet/sync1 (FF)
  Destination:          fifoGen[4].fifo/dataBuf_4_34 (FF)
  Source Clock:         clkNeighbour rising
  Destination Clock:    clkNeighbour rising

  Data Path: fifoGen[4].fifo/fullDet/sync1 to fifoGen[4].fifo/dataBuf_4_34
                                        Gate     Net
    Cell:in->out        fanout    Delay    Delay   Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q                2    0.591    0.526   fifoGen[4].fifo/fullDet/sync1 (
         fifoGen[4].fifo/fullDet/sync1)
     LUT3:I1->O             10    0.704    0.917   fifoGen[4].fifo/writeEnInt1 (
         fifoGen[4].fifo/writeEnInt)
     LUT3:I2->O             35    0.704    1.263   fifoGen[4].fifo/dataBuf_4_and00001
         (fifoGen[4].fifo/dataBuf_4_and0000)
     FDCE:CE                      0.555            fifoGen[4].fifo/dataBuf_4_0
    ----------------------------------------
    Total                        5.260ns (2.554ns logic, 2.706ns route)
                                         (48.6% logic, 51.4% route)

[...]
```

B.4 FPGA Implementation and Test

synth/TestEnv.syr

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

[...]

=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing Entity <TestEnv> in library <work> (Architecture <behaviour>).
WARNING:Xst:790 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/fpgaTest.
      vhd" line 105: Index value(s) does not match array range, simulation
      mismatch.
WARNING:Xst:790 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/fpgaTest.
      vhd" line 106: Index value(s) does not match array range, simulation
      mismatch.
WARNING:Xst:790 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/fpgaTest.
      vhd" line 111: Index value(s) does not match array range, simulation
      mismatch.
WARNING:Xst:790 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/fpgaTest.
      vhd" line 113: Index value(s) does not match array range, simulation
      mismatch.
Entity <TestEnv> analyzed. Unit <TestEnv> generated.

Analyzing Entity <sevseg> in library <work> (Architecture <Behavioral>).
INFO:Xst:1561 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/sevseg.vhd"
       line 118: Mux is complete : default of case is discarded
INFO:Xst:1561 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/sevseg.vhd"
       line 336: Mux is complete : default of case is discarded
WARNING:Xst:819 - "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/sevseg.
      vhd" line 156: One or more signals are missing in the process sensitivity
      list. To enable synthesis of FPGA/CPLD hardware, XST will assume that all
      necessary signals are present in the sensitivity list. Please note that the
      result of the synthesis may differ from the initial design specification.
      The missing signals are:
    <led3>, <leddp>, <led2>, <led1>, <led0>
Entity <sevseg> analyzed. Unit <sevseg> generated.

Analyzing Entity <routerFifo> in library <work> (Architecture <structure>).
Entity <routerFifo> analyzed. Unit <routerFifo> generated.

Analyzing generic Entity <fifo.2> in library <work> (Architecture <structure>).
  N = 5
  W = 35
Entity <fifo.2> analyzed. Unit <fifo.2> generated.

Analyzing generic Entity <tokenRing.3> in library <work> (Architecture <behaviour
    >).
  N = 5
  default = 3
Entity <tokenRing.3> analyzed. Unit <tokenRing.3> generated.

Analyzing generic Entity <tokenRing.4> in library <work> (Architecture <behaviour
    >).
  N = 5
  default = 12
Entity <tokenRing.4> analyzed. Unit <tokenRing.4> generated.

Analyzing generic Entity <fullDetector.2> in library <work> (Architecture <
    behaviour>).
  N = 5
Entity <fullDetector.2> analyzed. Unit <fullDetector.2> generated.

Analyzing generic Entity <emptyDetector.2> in library <work> (Architecture <
    structure>).
  N = 5
Entity <emptyDetector.2> analyzed. Unit <emptyDetector.2> generated.

Analyzing Entity <router> in library <work> (Architecture <struct>).
Entity <router> analyzed. Unit <router> generated.

Analyzing Entity <HPU> in library <work> (Architecture <struct>).
Entity <HPU> analyzed. Unit <HPU> generated.

Analyzing Entity <Xbar> in library <work> (Architecture <structure>).
Entity <Xbar> analyzed. Unit <Xbar> generated.

Analyzing generic Entity <fifo.1> in library <work> (Architecture <structure>).
  N = 10
  W = 35
Entity <fifo.1> analyzed. Unit <fifo.1> generated.

Analyzing generic Entity <tokenRing.1> in library <work> (Architecture <behaviour
    >).
  N = 10
  default = 3
Entity <tokenRing.1> analyzed. Unit <tokenRing.1> generated.

Analyzing generic Entity <tokenRing.2> in library <work> (Architecture <behaviour
    >).
  N = 10
  default = 12
Entity <tokenRing.2> analyzed. Unit <tokenRing.2> generated.

Analyzing generic Entity <fullDetector.1> in library <work> (Architecture <
    behaviour>).
  N = 10
Entity <fullDetector.1> analyzed. Unit <fullDetector.1> generated.

Analyzing generic Entity <emptyDetector.1> in library <work> (Architecture <
    structure>).
  N = 10
Entity <emptyDetector.1> analyzed. Unit <emptyDetector.1> generated.

Analyzing Entity <ClockDivider> in library <work> (Architecture <BEHAVIORAL>).
    Set user-defined property "CAPACITANCE = DONT_CARE" for instance <
        CLKIN_IBUFG_INST> in unit <ClockDivider>.
    Set user-defined property "IBUF_DELAY_VALUE = 0" for instance <
        CLKIN_IBUFG_INST> in unit <ClockDivider>.
    Set user-defined property "IOSTANDARD = DEFAULT" for instance <
        CLKIN_IBUFG_INST> in unit <ClockDivider>.
    Set user-defined property "CLKDV_DIVIDE = 5.0000000000000000" for instance <
        DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "CLKFX_DIVIDE = 1" for instance <DCM_SP_INST> in
        unit <ClockDivider>.
```

```
    Set user-defined property "CLKFX_MULTIPLY = 4" for instance <DCM_SP_INST> in
        unit <ClockDivider>.
    Set user-defined property "CLKIN_DIVIDE_BY_2 = FALSE" for instance <
        DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "CLKIN_PERIOD = 20.0000000000000000" for instance
        <DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "CLKOUT_PHASE_SHIFT = NONE" for instance <
        DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "CLK_FEEDBACK = 1X" for instance <DCM_SP_INST> in
        unit <ClockDivider>.
    Set user-defined property "DESKEW_ADJUST = SYSTEM_SYNCHRONOUS" for instance
        <DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "DFS_FREQUENCY_MODE = LOW" for instance <
        DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "DLL_FREQUENCY_MODE = LOW" for instance <
        DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "DSS_MODE = NONE" for instance <DCM_SP_INST> in
        unit <ClockDivider>.
    Set user-defined property "DUTY_CYCLE_CORRECTION = TRUE" for instance <
        DCM_SP_INST> in unit <ClockDivider>.
    Set user-defined property "FACTORY_JF = C080" for instance <DCM_SP_INST> in
        unit <ClockDivider>.
    Set user-defined property "PHASE_SHIFT = 0" for instance <DCM_SP_INST> in
        unit <ClockDivider>.
    Set user-defined property "STARTUP_WAIT = FALSE" for instance <DCM_SP_INST>
        in unit <ClockDivider>.
Entity <ClockDivider> analyzed. Unit <ClockDivider> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <sevseg>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        sevseg.vhd".
    Register <leddp<1>> equivalent to <leddp<0>> has been removed
    Register <leddp<2>> equivalent to <leddp<0>> has been removed
    Register <leddp<3>> equivalent to <leddp<0>> has been removed
    Found finite state machine <FSM_0> for signal <curan>.
    -----------------------------------------------------------------------
    | States             | 4                                              |
    | Transitions        | 4                                              |
    | Inputs             | 0                                              |
    | Outputs            | 8                                              |
    | Clock              | clk2 (rising_edge)                             |
    | Reset              | rst (positive)                                 |
    | Reset type         | synchronous                                    |
    | Reset State        | 11                                             |
    | Power Up State     | 11                                             |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <clk2>.
    Found 14-bit up counter for signal <count>.
    Found 4-bit register for signal <led0>.
    Found 4-bit register for signal <led1>.
    Found 4-bit register for signal <led2>.
    Found 4-bit register for signal <led3>.
    Found 1-bit register for signal <leddp<0>>.
    Summary:
  inferred   1 Finite State Machine(s).
  inferred   1 Counter(s).
  inferred  18 D-type flip-flop(s).
Unit <sevseg> synthesized.


Synthesizing Unit <tokenRing_3>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_3> synthesized.


Synthesizing Unit <tokenRing_4>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 5-bit register for signal <ring>.
Unit <tokenRing_4> synthesized.


Synthesizing Unit <fullDetector_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fulldetector.vhd".
    Found 1-bit register for signal <sync0>.
    Found 1-bit register for signal <sync1>.
    Found 1-bit register for signal <sync2>.
    Summary:
  inferred   3 D-type flip-flop(s).
Unit <fullDetector_2> synthesized.


Synthesizing Unit <emptyDetector_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        emptydetector.vhd".
    Found 5-bit register for signal <syncWriteP>.
    Summary:
  inferred   5 D-type flip-flop(s).
Unit <emptyDetector_2> synthesized.


Synthesizing Unit <HPU>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        hpu.vhd".
    Found 4-bit register for signal <selInt>.
    Summary:
  inferred   4 D-type flip-flop(s).
Unit <HPU> synthesized.


Synthesizing Unit <Xbar>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        xbar.vhd".
Unit <Xbar> synthesized.


Synthesizing Unit <tokenRing_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
    Found 10-bit register for signal <ring>.
Unit <tokenRing_1> synthesized.


Synthesizing Unit <tokenRing_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        tokenring.vhd".
```

```
    Found 10-bit register for signal <ring>.
Unit <tokenRing_2> synthesized.


Synthesizing Unit <fullDetector_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fulldetector.vhd".
    Found 1-bit register for signal <sync0>.
    Found 1-bit register for signal <sync1>.
    Found 1-bit register for signal <sync2>.
    Summary:
  inferred   3 D-type flip-flop(s).
Unit <fullDetector_1> synthesized.


Synthesizing Unit <emptyDetector_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        emptydetector.vhd".
    Found 10-bit register for signal <syncWriteP>.
    Summary:
  inferred  10 D-type flip-flop(s).
Unit <emptyDetector_1> synthesized.


Synthesizing Unit <fifo_1>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fifo.vhd".
    Found 350-bit register for signal <dataBuf>.
INFO:Xst:738 - HDL ADVISOR - 350 flip-flops were inferred for signal <dataBuf>.
     You may be trying to describe a RAM in a way that is incompatible with block
      and distributed RAM resources available on Xilinx devices, or with a
     specific template that is not supported. Please review the Xilinx resources
     documentation and the XST user manual for coding guidelines. Taking
     advantage of RAM resources will lead to improved device usage and reduced
     synthesis time.
    Summary:
  inferred 350 D-type flip-flop(s).
Unit <fifo_1> synthesized.


Synthesizing Unit <ClockDivider>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/Xilinx/mesorouter
        /ClockDivider.vhd".
Unit <ClockDivider> synthesized.


Synthesizing Unit <fifo_2>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fifo.vhd".
    Found 175-bit register for signal <dataBuf>.
    Summary:
  inferred 175 D-type flip-flop(s).
Unit <fifo_2> synthesized.


Synthesizing Unit <router>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        router.vhd".
    Found 175-bit register for signal <HPUout>.
    Found 175-bit register for signal <XbarOut>.
    Found 20-bit register for signal <XbarSel>.
    Summary:
  inferred 370 D-type flip-flop(s).
Unit <router> synthesized.


Synthesizing Unit <routerFifo>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        routerFifo.vhd".
WARNING:Xst:646 - Signal <fifoFull> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <fifoEmpty> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
Unit <routerFifo> synthesized.


Synthesizing Unit <TestEnv>.
    Related source file is "D:/Users/acb/Documents/DTU/Bachelor/src/mesochronous/
        fpgaTest.vhd".
WARNING:Xst:647 - Input <sw<7:6>> is never used. This port will be preserved and
     left unconnected if it belongs to a top-level block or it belongs to a sub-
     block and the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <btnOk> is never used. This port will be preserved and
     left unconnected if it belongs to a top-level block or it belongs to a sub-
     block and the hierarchy of this sub-block is preserved.
WARNING:Xst:646 - Signal <fifoFull> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <fifoEmpty> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <clkLockedOut> is assigned but never used. This
     unconnected signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <clkBufG> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <clk0Out> is assigned but never used. This unconnected
     signal will be trimmed during the optimization process.
    Found finite state machine <FSM_1> for signal <sendState>.
    -----------------------------------------------------------------------
    | States             | 4                                              |
    | Transitions        | 6                                              |
    | Inputs             | 2                                              |
    | Outputs            | 5                                              |
    | Clock              | clkW (rising_edge)                             |
    | Reset              | reset (positive)                               |
    | Reset type         | asynchronous                                   |
    | Reset State        | idle                                           |
    | Power Up State     | idle                                           |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 3-bit comparator equal for signal <fifoDest$cmp_eq0000> created at line
         94.
    Found 5-bit 3-to-1 multiplexer for signal <fifoRen>.
    Found 48-bit register for signal <numRecvd>.
    Found 48-bit 3-to-1 multiplexer for signal <numRecvdNext>.
    Found 8-bit adder for signal <numRecvdNext_0$addsub0000>.
    Found 35-bit comparator equal for signal <numRecvdNext_0$cmp_eq0000> created
        at line 151.
    Found 8-bit adder for signal <numRecvdNext_1$addsub0000>.
    Found 35-bit comparator equal for signal <numRecvdNext_1$cmp_eq0000> created
        at line 151.
    Found 8-bit adder for signal <numRecvdNext_2$addsub0000>.
    Found 35-bit comparator equal for signal <numRecvdNext_2$cmp_eq0000> created
        at line 151.
    Found 8-bit adder for signal <numRecvdNext_3$addsub0000>.
    Found 35-bit comparator equal for signal <numRecvdNext_3$cmp_eq0000> created
        at line 151.
    Found 8-bit adder for signal <numRecvdNext_4$addsub0000>.
    Found 8-bit adder for signal <numRecvdNext_5$add0000> created at line 155.
```

```
    Found 35-bit comparator equal for signal <numRecvdNext_5$cmp_eq0000> created
         at line 160.
    Found 8-bit 3-to-1 multiplexer for signal <numRecvdNext_5$mux0000> created at
         line 141.
    Found 8-bit 3-to-1 multiplexer for signal <numRecvdNext_5$mux0001> created at
         line 141.
    Found 8-bit 3-to-1 multiplexer for signal <numRecvdNext_5$mux0002> created at
         line 141.
    Found 8-bit 3-to-1 multiplexer for signal <numRecvdNext_5$mux0003> created at
         line 141.
    Found 8-bit register for signal <numSent>.
    Found 8-bit adder for signal <numSent$addsub0000>.
    Found 175-bit register for signal <recvBuf>.
    Found 10-bit register for signal <recvState>.
    Found 10-bit 3-to-1 multiplexer for signal <recvStateNext>.
    Found 3-bit up counter for signal <sendDest>.
    Found 3-bit register for signal <sendOrg>.
    Found 3-bit adder for signal <sendOrgNext$addsub0000> created at line 126.
    Found 31-bit up counter for signal <serial>.
    Summary:
  inferred   1 Finite State Machine(s).
  inferred   2 Counter(s).
  inferred 236 D-type flip-flop(s).
  inferred   8 Adder/Subtractor(s).
  inferred   6 Comparator(s).
  inferred  95 Multiplexer(s).
Unit <TestEnv> synthesized.

INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some
     arithmetic operations in this design can share the same physical resources
     for reduced device utilization. For improved clock frequency you may try to
     disable resource sharing.
```

```
=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 8
 3-bit adder                                           : 1
 8-bit adder                                           : 7
# Counters                                             : 3
 14-bit up counter                                     : 1
 3-bit up counter                                      : 1
 31-bit up counter                                     : 1
# Registers                                            : 175
 1-bit register                                        : 32
 10-bit register                                       : 15
 2-bit register                                        : 5
 20-bit register                                       : 1
 3-bit register                                        : 1
 35-bit register                                       : 90
 4-bit register                                        : 9
 5-bit register                                        : 15
 8-bit register                                        : 7
# Comparators                                          : 6
 3-bit comparator equal                                : 1
 35-bit comparator equal                               : 5
# Multiplexers                                         : 20
 1-bit 3-to-1 multiplexer                              : 5
 2-bit 3-to-1 multiplexer                              : 5
 8-bit 3-to-1 multiplexer                              : 10

=========================================================================
```

```
[...]

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : TestEnv.ngr
Top Level Output File Name         : TestEnv
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 31

Cell Usage :
# BELS                             : 4504
#      BUF                         : 7
#      GND                         : 1
#      INV                         : 11
#      LUT1                        : 43
#      LUT2                        : 855
#      LUT2_D                      : 1
#      LUT2_L                      : 15
#      LUT3                        : 701
#      LUT3_D                      : 15
#      LUT3_L                      : 11
#      LUT4                        : 2051
#      LUT4_D                      : 43
#      LUT4_L                      : 349
#      MUXCY                       : 133
#      MUXF5                       : 224
#      VCC                         : 1
#      XORCY                       : 43
# FlipFlops/Latches                : 3493
#      FDC                         : 725
#      FDCE                        : 2693
#      FDP                         : 1
#      FDPE                        : 41
#      FDR                         : 24
#      FDRE                        : 1
#      FDRS                        : 8
# Clock Buffers                    : 2
#      BUFG                        : 2
# IO Buffers                       : 28
#      IBUF                        : 7
#      IBUFG                       : 1
#      OBUF                        : 20
# DCMs                             : 1
#      DCM_SP                      : 1
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                     2937  out of   8672     33%
 Number of Slice Flip Flops:           3493  out of  17344     20%
 Number of 4 input LUTs:               4095  out of  17344     23%
 Number of IOs:                          31
 Number of bonded IOBs:                  28  out of    250     11%
 Number of GCLKs:                         2  out of     24      8%
 Number of DCMs:                          1  out of      8     12%
```

```
[...]

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 22.438ns (Maximum Frequency: 44.567MHz)
   Minimum input arrival time before clock: 12.253ns
   Maximum output required time after clock: 10.148ns
   Maximum combinational path delay: 2.675ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'DCM/CLKDV_BUF'
  Clock period: 22.438ns (frequency: 44.567MHz)
  Total number of paths / destination ports: 2245321 / 6248
-------------------------------------------------------------------------
Delay:               11.219ns (Levels of Logic = 25)
  Source:            fifoGen[3].fifo/dataBuf_3_1 (FF)
  Destination:       numRecvd_5_6 (FF)
  Source Clock:      DCM/CLKDV_BUF rising
  Destination Clock: DCM/CLKDV_BUF falling

  Data Path: fifoGen[3].fifo/dataBuf_3_1 to numRecvd_5_6
                                Gate     Net
    Cell:in->out      fanout    Delay    Delay   Logical Name (Net Name)
    ----------------------------------------  ------------
     FDCE:C->Q             1    0.591    0.455   fifoGen[3].fifo/dataBuf_3_1 (
          fifoGen[3].fifo/dataBuf_3_1)
     LUT4:I2->O            1    0.704    0.499   fifoGen[3].fifo/dataR<1>20 (fifoGen
          [3].fifo/dataR<1>20)
     LUT4_L:I1->LO         1    0.704    0.104   fifoGen[3].fifo/dataR<1>43 (fifoGen
          [3].fifo/dataR<1>43)
     LUT4:I3->O            1    0.704    0.455   fifoGen[3].fifo/dataR<1>75 (fifoOut
          <3><1>)
     LUT4:I2->O            1    0.704    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_lut<0> (
          Mcompar_numRecvdNext_3_cmp_eq0000_lut<0>)
     MUXCY:S->O            1    0.464    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<0> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<0>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<1> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<1>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<2> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<2>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<3> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<3>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<4> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<4>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<5> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<5>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<6> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<6>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<7> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<7>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<8> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<8>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<9> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<9>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<10> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<10>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<11> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<11>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<12> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<12>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<13> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<13>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<14> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<14>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<15> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<15>)
     MUXCY:CI->O           1    0.059    0.000
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<16> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<16>)
     MUXCY:CI->O           9    0.459    0.855
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<17> (
          Mcompar_numRecvdNext_3_cmp_eq0000_cy<17>)
     LUT3:I2->O            2    0.704    0.622   Mmux_numRecvdNext<5>7111 (N481)
     LUT4_D:I0->O          3    0.704    0.535   Mmux_numRecvdNext<5>12 (N89)
     LUT4:I3->O            1    0.704    0.000   Mmux_numRecvdNext<5>7 (numRecvdNext
          <5><6>)
     FDC:D                      0.308            numRecvd_5_6
    ----------------------------------------
    Total                      11.219ns (7.694ns logic, 3.525ns route)
                                        (68.6% logic, 31.4% route)

=========================================================================
Timing constraint: Default period analysis for Clock 'display/clk2'
  Clock period: 3.108ns (frequency: 321.750MHz)
  Total number of paths / destination ports: 2 / 2
-------------------------------------------------------------------------
Delay:               3.108ns (Levels of Logic = 1)
  Source:            display/curan_FSM_FFd1 (FF)
  Destination:       display/curan_FSM_FFd2 (FF)
  Source Clock:      display/clk2 rising
  Destination Clock: display/clk2 rising

  Data Path: display/curan_FSM_FFd1 to display/curan_FSM_FFd2
                                Gate     Net
    Cell:in->out      fanout    Delay    Delay   Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q             19    0.591    1.085   display/curan_FSM_FFd1 (display/
          curan_FSM_FFd1)
     INV:I->O              1    0.704    0.420   display/curan_FSM_FFd2-In1_INV_0 (
          display/curan_FSM_FFd2-In)
     FDR:D                      0.308            display/curan_FSM_FFd2
    ----------------------------------------
    Total                      3.108ns (1.603ns logic, 1.505ns route)
                                        (51.6% logic, 48.4% route)
```