Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-MS-2014-05

M.Sc. Thesis

Physical design of a 3D router: reducing the number of vertical connections and enabling asynchronous operation

Milovan Vasić

Abstract

With the use of multi-core architectures, the Network-on-Chip (NoC) became an important research topic. The most important benefit of a NoC compared to a communication bus is that it is scalable. The heart of the NoC is the router, which provides the communication between different computational units. This component is highly suitable to be a 3D component, which means that the connection can go into a vertical direction. This way the NoC is extended, with the same area footprint. This thesis describes the physical design of the 3D router, where various design problems are solved. An existing router architecture is used as a start-point. One of the problems which this thesis is trying to solve, is the reduction of the number of needed data lines. This is especially useful for the vertical data lines, which are implemented with Through Silicon Vias (TSVs). A TSV has a large footprint compared to a transistor, which means it takes up a lot of chip area. This results in increased cost. The reduction of the data lines is accomplished by the serialization of the data. It is determined that the best serialization ratio is 4. The 3D router is also adjusted for asynchronous operation. This is accomplished with the use of FIFOs, two-flop synchronizers and Gray Encoders.

Physical design of a 3D router: reducing the number of vertical connections and enabling asynchronous operation

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Microelectronics

by

Milovan Vasić
born in Vlissingen, the Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Physical design of a 3D router: reducing the number of vertical connections and enabling asynchronous operation"** by **Milovan Vasić** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 9 May 2014

Chairman:

Prof.dr.ir. A.J. van der Veen

Advisors:

Dr.ir. T.G.R.M. van Leuken

Dr.ir. N.P. van der Meijs

Committee Members:

Dr.ir. M.R.C.M. Berkelaar

Dr.ir. A.J. van Genderen

Abstract

With the use of multi-core architectures, the Network-on-Chip (NoC) became an important research topic. The most important benefit of a NoC compared to a communication bus is that it is scalable. The heart of the NoC is the router, which provides the communication between different computational units. This component is highly suitable to be a 3D component, which means that the connection can go into a vertical direction. This way the NoC is extended, with the same area footprint. This thesis describes the physical design of the 3D router, where various design problems are solved. An existing router architecture is used as a start-point. One of the problems which this thesis is trying to solve, is the reduction of the number of needed data lines. This is especially useful for the vertical data lines, which are implemented with Through Silicon Vias (TSVs). A TSV has a large footprint compared to a transistor, which means it takes up a lot of chip area. This results in increased cost. The reduction of the data lines is accomplished by the serialization of the data. It is determined that the best serialization ratio is 4. The 3D router is also adjusted for asynchronous operation. This is accomplished with the use of FIFOs, two-flop synchronizers and Gray Encoders.

Acknowledgments

I want to thank Rene van Leuken for finding me a very interesting topic for doing my master thesis. I wanted to do something in chip design, especially in 3D chip design. This was the perfect opportunity to extend my knowledge on this topic. The road to accomplish this was not easy and it took me some time, but at the end I succeeded. Thank you Rene for the countless times I bumped in your office for questions, and you helped me. This meant a lot to me.

A person who joined later was Michel Berkelaar. I want to thank you for all your support during my master thesis. Not only the educational support you have given me, but especially mental support which I needed from time to time. Because your background lies in the industry, you could teach me a lot of stuff which were hard to find in textbooks. Thank you very much for this.

Sumeet, using your original router design, I had a lot of questions for you. For every question you always took the time to explain it to me. Thank you kindly for that! Radhika, even when you're abroad, and working for a big firm, you have found time to answer me a couple of questions I fired at you. It helped me a lot and thanks for that.

Of course all other teachers, staff members who I might have forgot to mention, thank you for all your support. Also you were a big part of helping me on the way finish my master thesis.

Great thanks goes out to my family, who have always supported me. Without them, I would never have managed to accomplish what I have today. Both of my parents, and both of my brothers, thank you for everything!

Milovan Vasić
Delft, The Netherlands
9 May 2014

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

1.1.1 Background

In current IC technology several problems exist regarding to chip design. They are related to the desire for performance increase, which includes higher speed, lower power dissipation and smaller area. Speed is related to computational performance. The usual method to increase computational performance was to use more transistors. One way to accomplish that was to shrink the channel length (transistor size), which meant more transistors could be put on the same chip area and the computational performance would increase. It also means that the transistors are getting faster and their capacity lower, which also results in an increase in computational performance. A problem with this method is that the channel length of the transistor is already reaching the 10 nm border where various effects come into play. These effects include tunneling and energy quantization [2], which significantly increases the complexity of the operation of the transistor. Another way to increase the number of transistors is to increase the chip area, so more transistors can be put on the chip. The problem with this method is that with the increase of the chip area, the possibility of defects occurring also increases. This means that after manufacturing, the number of working chips will be reduced (low yield), resulting in increased cost.

The demand for increased performance in a chip caused engineers to find different solutions. A trend that could be observed was that engineers began to use multi-core chip design. An advantage of multi-core architectures is that they are suitable for parallel computing, which improve computational performance in general. This type of chips are known as *chip-multi-processor (CMP)*. The communication between the different cores is done by communication buses. With the increase in the number of used cores, the complexity of the buses increases, resulting in long and complex interconnections [3]. This causes the delay, power dissipation and occupied chip area to become larger. The delay results in decrease of the throughput. Because more cores cause problems for the communication buses, this system is not suitable for scalability. Because transistors were shrinking much faster then wires, wires became the larger component of the two, resulting in longer delays and increased power dissipation [4]. The connection in a chip was usually based on point-to-point connections. With the increasing complexity of chips, it became increasingly hard to obey the real-time constraints in point-to-point connections [5] which were needed to let the design work properly.

1.1.2 Network-on-Chip

A solution for the communication problems, which is extensively researched, is the *Network-On-Chip (NoC)*. A NoC provides the communication between different computational units, like CPU's. It consists of routers, which are connected to each other in a mesh, and where different computational units are connected to them through a *Network Interface (NI)*. Using a NoC instead of a communication bus solves the complex wiring problem. Because the computational blocks are only connected to their router, and since the router determines the communication protocol, no long wiring is needed, and no complex real-time constraints need to be obeyed. An advantage of a NoC is that it is scalable, which means more routers could be added together with their computational units, without degradation of performance. This NoC can be extended with 3D routers, enabling 3D chip design which is done by the vertical connections of the 3D router. With a 3D router, which serves as a 3D component, multiple meshes can be stacked on top of each other, increasing the computational performance without increase of the chip area. Two meshes stacked on top of each other are shown in Figure 1.1.

There already is a 3D router design by S. Kumar [6] which can serve as a 3D component. The router had initially 5 ports (north, south, east, west, local). This was extended by two additional vertical ports (up, down) which made this a 3D component. Every port can send and receive 37-bit data flits. For this design a RTL model is available, which simulates correctly, but there is no physical design available. In order to be able to use this 3D component for 3D chip design, some design problems like timing, placement and physical implementation of the vertical connections should be solved. After this, physical design can be done.

1.1.3 Design problems 3D router

One of the design problems in the router design by S. Kumar, is that it uses 37-bit wide data lines for all the input/output ports. This means that every port consists of 37 lines for the input and 37 lines for the output. On every clock cycle of the router, one bit is sent through a data line. Because this data line has a very small delay compared to the clock cycle on which the router is operating, a bit can be sent at a much faster interval, compared to the clock cycle of the router. In this way the capacity of a data line is more efficiently utilized. If the interval for sending a bit through a data line would be decreased, more bits could be transported, which means fewer data lines would be needed in order to transport an equal amount of bits.

Another problem in the router design by S. Kumar is that it only works in one clock domain. This means that when used in an unmodified NoC configuration, all the routers should operate on the same clock frequency. This is hard to achieve, especially between two routers sitting on different tiers. A solution has to be found which enables the operation in different clock domains.

The physical design problems that arise when using the router design of S. Kumar, are the need for *Trough Silicon Vias (TSVs)* for every data line in the vertical direction. As explained earlier, there are 37 input and output data lines in each direction. Because a TSV has a large footprint compared to a transistor, a lot of chip area will be needed

in order to accommodate all the TSVs which will result in cost increase. If the number of needed TSVs could be reduced, less area would be taken by them, resulting in lower costs.

This thesis is trying to solve the above mentioned problems by using a new design methodology. The design problems of the router by S. Kumar which this thesis is trying to solve are:

- Low utilization of the capacity in the data lines
- Communication between multiple clock domains
- Large footprint of needed TSVs

1.2 Contributions

This thesis concentrates on the reduction of the needed data lines in the vertical direction in order to improve the throughput of the data lines, and to reduce the number of needed TSVs because of their footprint. The technique which is used, is serialization/de-serialization. For this an existing design of a serializer/de-serializer is used [5] and implemented in the 3D router. This design makes use of two clocks, which ensures that the serialization and de-serialization is done without the need of additional clock cycles. Some timing problems are solved in order to ensure valid operation of the serializer/de-serializer. To enable operation in multiple clock domains, the FIFO from the router, which consists of a two-port register file, is used together with two-flop synchronizers and Gray encoders [7].

A physical design is made for the adjusted 3D router. The design of choice is an ASIC semi-custom design, which consist of a standard cell library and some macro's. The standard cell library is composed of the 45 nm Nangate Open Cell Library. For the FIFO, macro blocks are used, which consist of 55 nm Faraday memory blocks (two-port register files). These blocks are adjusted for the 45 nm grid. The EDA tools which are used for physical design are Synopsys Design Compiler for synthesis and Cadence SoC Encounter for place&route. For the TSV a macro is made, which is composed of a RTL model, an abstract layout model (LEF file) and a timing model (Liberty file) which is based on the work of R. Jagtap [8]. Because it is intended to let the router operate on as high a clock frequency as possible, some timing optimization steps are performed.

At the end, two asynchronous routers with a serializer and de-serializer in between are connected through their vertical interconnections, which forms a basic NoC. To test the operation of this design, an injector with a large test coverage is used, made as a vhdl testbench, which emulates a real computational unit like a CPU core (ARM, Micro-Blaze, X86, etc.). When connecting several 3D routers on all sides, a complete mesh is made which looks like Figure 1.1. In this thesis only two vertically connected routers are tested, because this forms the most complex communication where the serialized data lines are tested, together with the asynchronous operation of the two routers. The floorplan which is used is kept the same, where the number of TSVs are changed, dependent on the serialization ratio of the data in the vertical direction. Also different memory sizes are tested to see how they impact the area of the chip. In the

end there is an overview of different results, like the throughput, latency, complete chip area, memory area and TSV area, together with the timing results. This way it can be determined how the different topologies and serialization factors influence the occupied chip area and performance.

The significant contributions of this thesis include:

- Reduction of data lines in the vertical direction of a 3D router
- Enable operation of a 3D NoC in multiple clock domains
- Physical design of a 3D router
- Timing optimization in physical design of a 3D router

1.3 Thesis Organization

The thesis is organized in the following way:

Chapter 2 will describe some background information, compare similar previous implementations and discuss what will be improved in this implementation.

Chapter 3 will describe the design steps on how to make a physical design of a 3-dimensional router with 7 ports, where the 2 vertical ports are made with the use of TSVs. First the architecture of the router [6] and then the implementation of it will be explained. The implementation of the memory and TSVs into the design will be discussed with the help of the used design flow and the needed EDA tools.

Chapter 4 will describe the design of the serializer and de-serializer [5], together with the implementation into the design. Some solutions to timing problems will be given at the end of this chapter.

Chapter 5 will describe the design of an asynchronous router using FIFO's, two-flop synchronizers and Gray encoders [7].

Chapter 6 will describe the performance of two vertically connected asynchronous 3D routers where the data which is flowing in the vertical direction is serialized. Also the timing results will be given for 1 3D router with different serialization ratios.

Chapter 7 includes the concluding remarks and recommendations for future work.

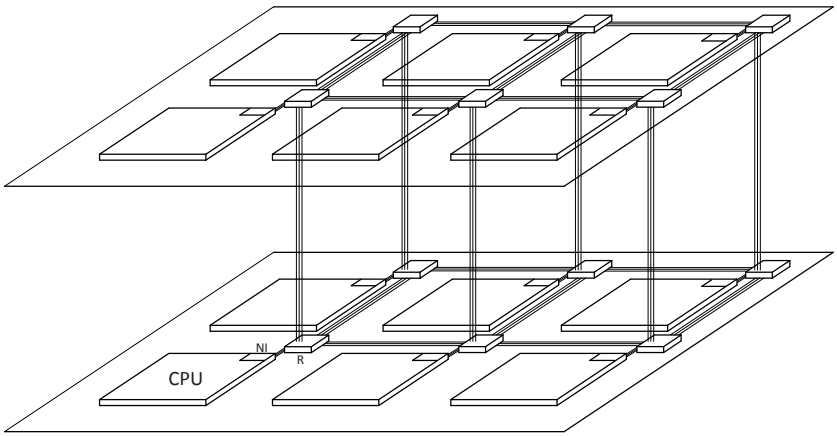


Figure 1.1: 3D NoC with CPU, network interface and 7-port router

2 Background

The design of a complete three dimensional chip is a long and complicated process. For an engineer it is important to divide the design into several manageable parts and decide on how to implement them. Each part can be implemented in a different way, which means that a good balance has to be found between different implementations that suit all the requirements with the lowest cost.

To understand the 3D router design and the choices made here, it is important that some background information is given on this subject. This chapter gives some background information about the implementation of 3D chip design. First the most important parts in 3D chip design will be described, after which a link will be made to the 3D router design. A complete workflow for this design will be given at the end of this chapter.

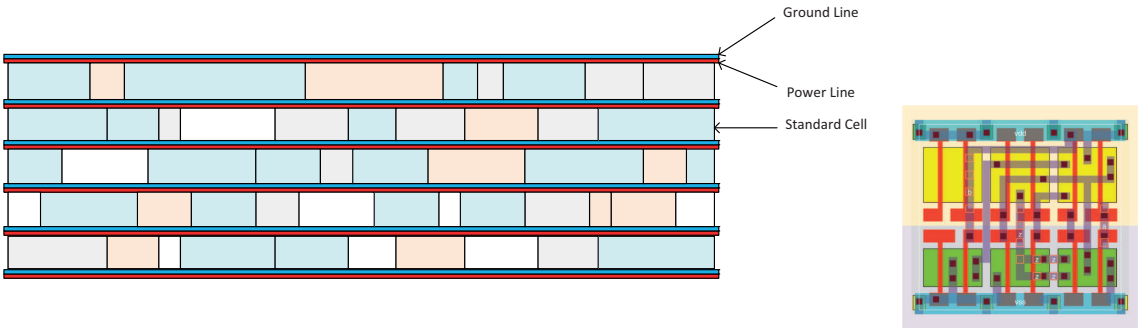
2.1 Design and Components

This section describes the different implementation parts of 3D IC design and the needed components. First a general description is given on the implementation of IC design. After that the standard cell library, memory and TSV will be discussed. At the end some 3D implementations will be described and compared. This information mainly serves to provide more insight in the complete process of the 3D router design, and to understand why some decisions are made.

2.1.1 IC Design Methodology

An engineer has to decide which technology he wants to use. Whether it be an *ASIC (Application-Specific Integrated Circuit)* design or a *FPGA (Field-programmable gate array)* design, a full-custom design or a semi-custom design. This choice will depend on the constraints set before. An ASIC design is highly suitable for designs with a specific function. The 3D router has such a specific function, which makes it perfect for ASIC design. An ASIC design can be full-custom or semi-custom.

A full-custom design is the fastest design, but it is also the most complicated one, and the design time is the longest. In this type of design all of the transistors are shaped and placed by hand. With the access to the transistor design itself, which is the main building block of any digital circuit, a designer has the freedom to chose different layouts for different small circuits, and to chose the most efficient placement. This also means the designer should keep in mind the different design rules. Design rules specify different constraints like minimum-width and minimum-spacing requirements between objects on the same or different layers [4]. The drawback of full-custom design is, that it is time consuming, which translates automatically into high costs. This type of



(a) Standard Cell rows

(b) Standard Cell (XOR2) [9]

Figure 2.1: ASIC semi-custom design

design is mainly suitable for specific performance-critical modules which can be reused in different designs. Another important area where full-custom design is used, is for developing standard cells which are used in semi-custom design, or for analog blocks in analog design.

A semi-custom design is another possibility for designing ASIC. There is some performance degradation which can be accepted, while at the other hand the design time is reduced. An semi-custom design consists primarily of standard cells and macro blocks. Standard cells are usually available from different vendors, who make a standard cell library. A standard cell is a cell with a predefined height and adjustable width. This cell consist of a logical gate, i.e. NOR, NAND, INV, DFF, etc. At the top of the cell, there is a power line, where at the bottom of the cell, there is a ground line. The standard cells are made full-custom, to have the best possible characteristics. The designer who is using these standard cells, doesn't have to worry about the inside of the cells. The most important characteristics are the cell delay and power dissipation. These cells are placed in rows, with power lines at the top of the row, and ground lines at the bottom. In Figure 2.1a different rows can be observed with power/ground lines and standard cells. In Figure 2.1b a layout of a XOR2 standard cell can be observed. Because of it's efficient use and good performance, semi-custom design is highly suitable for the 3D router design.

Macro blocks form another important part of a semi-custom design. Using standard cells is attractive for random logic functions, but it's quite inefficient for more complex parts like memory, DSP's, etc. [4]. These parts can be designed in a more efficient way, by using a specific structure that is suitable for it. This ensures the customized implementations will outperform standard cell design by a wide margin. There are two types of macro cells, the hard macro cell and the soft macro.

The hard macro is a module with a given functionality and a predefined physical design [4]. An ASIC designer should not worry about the inside of this block, but only about the input/output characteristics and power consumption, just like with a standard cell. The advantage of a hard macro is, that the design itself is very efficient, and that a block can be reused more times. A disadvantage would be that when a hard

macro is designed for a particular technology, it will only work with that technology. It is very hard to port a hard macro to another technology. An area where a macro block is highly usable is when making a design with a regular structure. Here the same components are used over and over again, and put into a regular structure. A good example of this is a memory block. When designing a memory block, there is a basic building block, which can be extended into a regular structure for a desired memory size. Generators are usually made for this purpose. Such a generator will create a macro block with the help of some input arguments. For example, these arguments can be the desired address size or word size.

The soft macro cell is a module with given functionality, but without specific physical implementation. The placement and wiring will determine the timing of that block, which means that they can change every time a new synthesis and placement&routing is ran. Compared with a hard macro, it has a disadvantage that it doesn't use full-custom design, which is important when using high performing blocks like memories. The advantage is that it's not technology specific, and that it can be ported to other technologies quite easily.

2.1.2 Standard Cell Library

For the logic in the 3D router design, a standard cell library has to be chosen. These libraries are usually designed by IC vendors. They develop their own cells, characterize them, and sell them to other companies for implementation. Libraries are made with the help of a *Process Design Kit (PDK)*. A PDK is a bundle of files which describes different aspects of transistors needed to design standard cells. These files consist of design rules, simulation models for transistors and layout information for transistors. With these files, standard cells can be made, and in the end a complete library of these cells can be constructed.

The case here is, to chose the best possible library which will support the different aspects of the 3D router design. The most important aspect the chosen library should have, is support for 3D design. This means it should have support for TSVs. Another important aspect is that it should be made in a small process technology. The third aspect would be that it is available for free. One Standard Cell Library that was qualified for the above mentioned conditions, was the Nangate Open Cell Library. The purpose of this library is to provide research, testing, and exploring of different EDA flows [10]. The library is generated with the Nangate Library Creator, and the 45 nm FreePDK Base Kit from *North Carolina State University (NCSU)*, and it is characterized with the *Predictive Technology Model (PTM)* from Arizona State University. The only drawback of this library is, that it has not industrial grade performance, because it was generated using a non-optimized Open PDK [10]. This is not a real problem, because the most important aspect of this thesis is to make a functional 3D router design, in real technology, and that it should work. The design flow and the different optimization steps in such a physical implementation are one of the main aspects of this thesis.

The support for 3D design is not directly incorporated into the Nangate Standard Cell Library, but it is supported by the PDK which is used to make this library. This PDK contains the technology library, which supplies tech files, display resources, design

rules and scripts to permit layout design and rule checking for a generic 45 nm process [11]. Because in this design flow primarily the Standard Cell Library will be used, only the relevant part of the PDK will be discussed, which is the TSV implementation.

In Section 2.1.4 some background information on TSV will be given, and different technologies will be discussed, and in the end, a comparison will be made with the one from the NCSU PDK.

2.1.3 Memory

For every input port of the 3D router, a memory block is needed. This memory block will be used as an input buffer. It should obey the different constraints needed for the input buffers, like word depth, and address length. Later on in Chapter 5 it will be explained how this memory can be used to separate two clock domains. The available memory is a 55 nm synchronous, two port register file from Faraday. In section Section 3.3 the exact specs will be given and explained in the context of the complete 3D router design. The model is given in several files. An HDL file, a LEF file and a liberty file are all needed for the simulation, place&route and timing information.

The only problem with this memory block is, that it's based on 55 nm technology, instead of 45 nm technology like the Nangate Standard Cell Library. The reason for using 55 nm technology is because this was the closest available memory technology compared to the 45 nm of the Nangate Standard Cell Library. This was solved by some modifications explained in chapter Chapter 3.

2.1.4 Through Silicon Via

A TSV is a vertical connection which is made into silicon. First a hole is etched into the silicon, and then it's filled with a metal like aluminum, tungsten, wolfram, etc. There are basically three steps which are used for the complete 3D process: via drilling and filling, wafer thinning and backside metallisation [12]. The positive side of TSV's is that it enables vertical interconnections, and allows the stacking of different dies. This enables the reduction of the chip footprint, and at the same time improves the performance due to the vertical interconnections.

To be able to use TSVs in 3D IC design, first a better understanding of the TSV should be obtained. In this section some background information will be given on this subject, like size, used materials, placement, and reliability. At the end a comparison will be given with the TSV model from the NCSU FreePDK, which will be used as a reference design. Later on in chapter Chapter 3 the implementation of the TSV will be discussed, which will be based on the work by R. Jagtap [8].

2.1.4.1 Size

A TSV can have different diameters and different pitch. The smaller the diameter is, the smaller the pitch, and the denser the design. This means also the larger the diameter, the larger the pitch and the more sparse the design. The problem with TSVs is that it cannot shrink the same way like transistors do. This is why TSVs are far bigger than transistors. On of the reasons the TSV cannot be shrunk under a certain

size is if the diameter of a TSV is shrunk, the wafer thickness should also be smaller, because of via filling reasons [12]. [12] states that when using TSV's with the diameter of 10 nm and less, the density can go high as 10000 TSV/mm^2. Nevertheless, very small diameter TSVs have been designed where 4 nm wide TSV have been patterned with lithography using 3.2 μm thick photo-resist [13].

A new technology which enables TSVs with a diameter in the order of 50 nm is monolithic 3D integration. This technology splits NMOS and PMOS transistors in each standard cell into two device tiers. Thanks to the extremely small size of the inter tier vias, ultra fine-grained vertical integration of devices and interconnects are enabled [14].

2.1.4.2 Placement

The placement of TSVs on a chip determines the reliability and speed. For example, a regular placement of TSVs improves the exposure quality of the lithographic process and therefore improves the yield [15]. On the other end, the *keep-out-zone (KOZ)* is an area outside of the TSV, where no transistor can be because of reliability issues. But because of this, the pitch of the TSV's becomes larger, so does the area which will be covered by TSV's and KOZ. The reliability issues related to KOZ are explained in Section 2.1.4.3. R. Jagtap compares several TSV placement topologies, and states that area and performance wise, the shielded and isolated topologies are preferred [8].

2.1.4.3 Reliability

Reliability is an important aspect in developing TSV's to enable maximum performance and high current demands trough the stacked package. [16] states that there are two critical interfaces on high density TSV with diameters below 10 μm. The first one is the transition between TSV and first metal level of *Band End of Line (BEoL)*–which is a thin metallization having a thickness below 0.25 μm. The second one is the transition between TSV and backside redistribution level–a thick metallization of about 1 to 5 μm. A thick metal process should be expected to increase the EM robustness, based on current density considerations. This is not the case due to void nucleation occurring right at TiN interface of TSV instead of opposite SiN capping layer. Because of this, any thickness increase of metal level in order to increase the electromigration robustness, should be considered carefully, because of void nucleation interface and void growth.

[17, 18] states that there are three levels of partitioning granularity. There is core level integration, block level integration, gate-level integration and transistor level integration. Current IC technologies support TSV diameters down to 5 μm. Alignment precision is very important in stacking tiers.

When using TSV's, there is a redundancy problem. If one TSV fails in a chip, the complete chip stops working, which means such a failure can cause an increase in cost and decreases in yield as the number of dies to be stacked increases [17, 18]. With one redundant TSV allocated to one TSV block, the proposed structure leads to 90% and 95% recovery rates for TSV blocks of size 50 and 25, respectively.

Presently, most 3-D IC processes require each tier to be less than 100 μm thick [15]. There is a significant difference in thermal expansion between Copper and Silicon,

which can cause significant stresses at the interface during thermal processing [19, 20]. These thermal stresses can cause mechanical stresses in the channels of the n-MOS and p-MOS devices, which will result in an increase or decrease of the drain current of the MOS devices. Because of this, a KOZ can be formed around the TSV, to minimize this effect. The KOZ for an TSV is defined as an area around the TSV where the variation in the drain current is below a certain threshold. Equation (2.1) shows the connection between the KOZ area and the TSV diameter.

$$\frac{KOZ_1}{KOZ_2} = \left(\frac{\phi_1}{\phi_2}\right)^2 \tag{2.1}$$

When scaling the TSV diameter, the KOZ area reduces quadratically. Because of this, it is very important do reduce the TSV diameter.

[21] used a Bosch-process for making TSV's with a depth of 40 μm and a diameter of 5.2 μm. If a transistor is placed parallel from the TSV, the effect is enhanced, and if it is placed perpendicular, this effect gets smaller. The KOZ for a large matrix of TSVs is over 200 μm for analog circuits and 20 μm for digital circuits.

Coefficient of thermal expansion (CTE) mismatch between the conducting metal in TSV and silicon substrate can generate thermal stress inside and around TSVs [22]. For TSV interfacial crack, larger landing pad size is not beneficial. A landing pad is used to connect a TSV to a metal layer in a tier. The smallest defects occur when the TSVs are placed 90°, because the tensile and compressive stress from each aggressor TSV cancels out at the victim TSV location. There is a high angular dependency in a small pitch region. If the pitch exceeds 15 μm, angle impact is almost negligible. Array type placement causes the lowest stress. It is best to use block placement in the design. This agrees to the highest degree with the findings of R. Jagtap, where the bundle topology and shielded topology (same as bundle, only with ground TSVs between signal TSVs because of capacitive coupling issues) are chosen as the best one [8].

2.1.4.4 Comparison with NCSU FreePDK TSV

Table 2.1 shows the different dimensions of the metal layers in a tier, which are part of a stacked 3D IC design. This is illustrated in Figure 2.2. In this process, the TSV is created out of different metal layers. All the metal layers stacked on top of each other, will form a vertical connection. Every metal layer has it's own minimum dimensions. But because all these layers are stacked on top of each other, the layer with the largest dimension will determine the dimension of the complete TSV. In this case this dimension pertains to the gate oxide, substrate and back metal, which is 6000 nm or 6 μm. Compared with the TSV sizes in Section 2.1.4.1, 6 μm is acceptable. Also the TSV size which is used in [8] of 4 μm is of comparable magnitude. Only the technology from [13] and [14], which uses TSV's with a diameter of 10 nm and less are far smaller. But these technologies are not suitable for this router design, because of the lack of standard cells which support this kind of implementation.

The material used for the NCSU FreePDK TSV is not important in this design process. The only thing that matters, is that the dimensions of the TSV are known and obeyed. Because of this, the will be no further discussion about the TSV material.

For the placement of TSVs, several topologies can be used. In section Section 2.1.4.3 it is stated that an array type of placement is the most suitable for high reliability. Performance wise this is good enough to be used in this particular design. When using the TSV dimensions from the NCSU FreePDK technology, the operation should be acceptable. Implementing the KOZ shouldn't be a big problem, because of the used placing of TSVs. More details on the TSV and KOZ area will be given in chapter Chapter 6.

Table 2.1: Metal information for 1 tier [1]

Name	**Pitch (Width/Space) (nm)**	**Thickness (nm)**	**Via dimension (nm)**
Top Metal	1600 (800/800)	1000	-
TM Cap Oxide	-	1000	800
ILD 9	-	2000	800
Global (9-10)	1600 (800/800)	2000	-
ILD (7-8)	-	820	400
ThinGlobal (7-8)	800 (400/400)	800	-
ILD 4-6	-	290	140
Semi-global	280 (140/140)	280	-
ILD 2-3	-	120	70
Intermediate (2-3)	140 (70/70)	140	-
ILD 1	-	120	65
Metal 1	130 (65/65)	130	-
Poly-Dielectric	-	85	65
Poly	125 (50/75)	85	-
Gate Oxide	-	200	**6000**
Substrate	-	40000	**6000**
BM Cap Oxide	-	200	**6000**
Back Metal	1600 (800/800)	1000	-

2.1.5 3D Implementation

When having a multi-core design using a NoC as an interconnect, a three dimensional implementation could be very suitable. First a decision has to be made how to implement this 3D technology. As stated in Section 2.1, the use of TSVs is the solution. There are several ways of implementing these vertical interconnections. All of them are based on at what level the TSV will be implemented. A description of the most used implementations is given:

3D-SIP [23] The most straightforward solution is to stack different dies on top of each other, and to make the interconnections with bond wire at the peripheral pins. This defined as a *3D system in a package (3D-SIP)*. The problem with this kind of implementation is that the number of available interconnects (pins) is too low for an complicated and fast design. Because of this reason, this implementation is not used a lot in current designs.

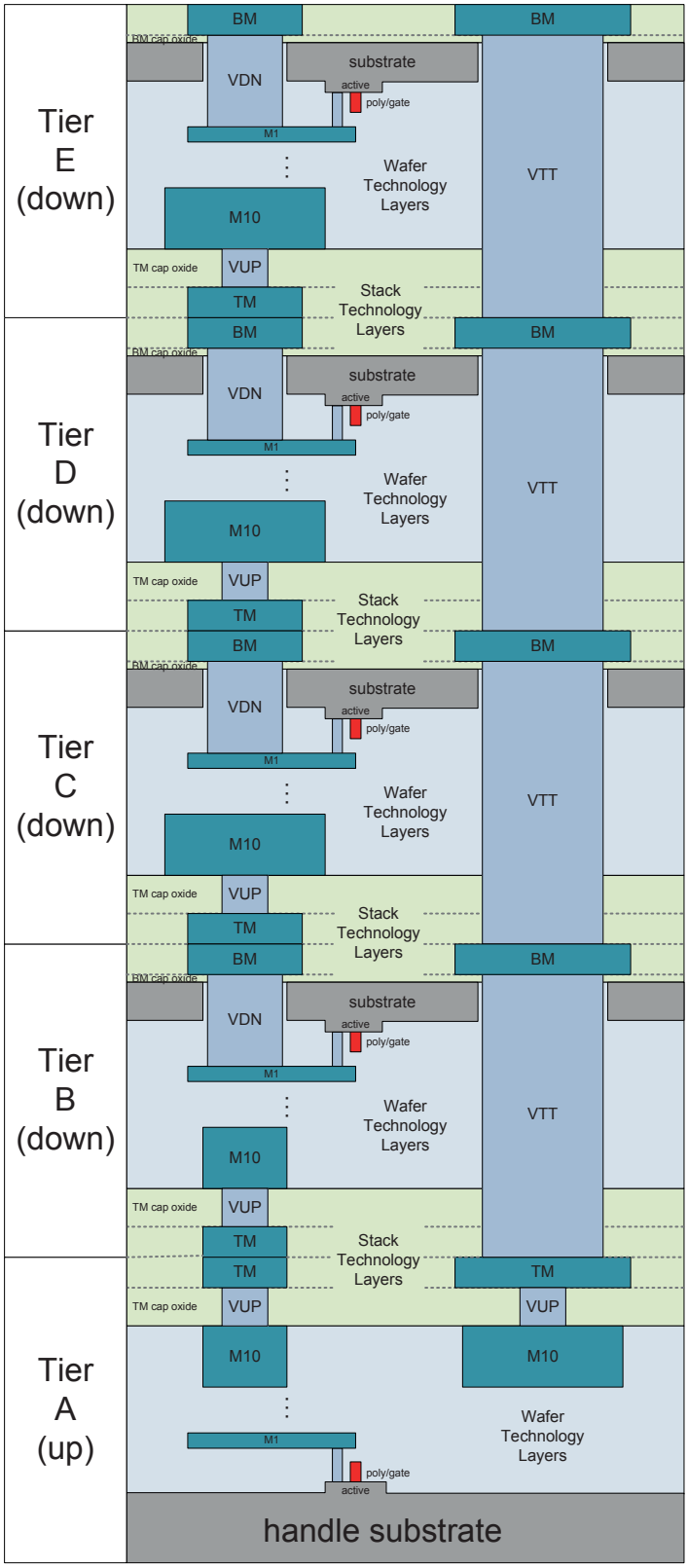


Figure 2.2: Metal Cross-Section Diagram [1]

3D-SoC [23] Another implementation is with the use of internal vertical interconnections. This implementation is based on *Trough Silicon Via (TSV)*, which is an interconnection with low capacity and resistance. The the interconnections are made on block level. Different blocks are interconnected with each other with TSV's and the number of available interconnections is much higher than the 3D-SIP implementation. This implementation is defined as *3D System on a Chip (3D-SoC)*.

3D-IC [23] 3D interconnection at the lowest level (transistor level) is called *3D-integrated circuit (3D-IC)*. Here the interconnects are made locally, where the global interconnections stay 2D. In this technology, the number of 3D interconnections per unit area are the highest. This implementation is used only in very high performance designs.

When comparing these different 3D implementations, the 3D-SoC implementation seems to be the most suitable for the 3D router design. A 3D NoC consists primarily of 3D routers which can be seen as blocks, and have those two vertical interconnections made with TSVs. With the 3D-SoC implementation, a mesh of 3D routers (and attached CPU's) can be made, which is shown in Figure 1.1.

2.2 Approach

Now that most components have been described, a way has to be found to combine them, and design a complete digital system, in this case a 3D router. The fact is that there are a lot of different ways to design a complete system, but it is important to find one which will suite a particular design the best possible way and also be developed in an efficient way.

The first thing that has to be decided upon, is the architecture of the system. The architecture can determine for example if the design will be a single-core design or a multi-core design. It can determine if it will be a single clock-cycle design or a multi clock-cycle design.

Right after this comes the implementation. As stated in previous sections, the implementation is very broad, because a design can be very complex and consists out of several specialized blocks. These block can be digital, analog, low power, high performance, they can be placed on the same die or they can be placed on different dies. The digital system can be improved by extending the architecture, which will use more transistors, which in turn will make the design larger and more complicated. Another way to improve the design is to use multiple cores, and let them communicate with each other. These cores can be similar, or they can be different cores with different functions.

2.2.1 Motivation

The motivation for this type of design comes from several problems that exist in current chips. The first one is the communication problem, where all the communication is done by direct interconnection. This can give some performance overhead. A trend

that can be observed today is that digital systems get multiple cores or many different digital blocks on the same die. This takes up a lot of area, and it is even hard from a technological perspective to make these different blocks on the same die. Also the large amount of complex wiring is causing some problems.

To solve the first problem, a NoC is introduced. A NoC gives support for multi-core designs, and multi-block designs. It enables a network of different cores that all communicate with each other in a efficient way where the large amount of complex wiring is reduced. Also it enables scalability, which enables adding or removing different digital cores and/or blocks. This can be of interest when extending or changing the design.

Because of the many different cores and blocks on a die, the die can get very large. It should be possible to stack the different cores on top of each other. This is made possible with the introduction of TSVs, which are vertical interconnections, and enable the interconnection in three dimensions. In this way, the chip area can be reduced and the communication can be improved because of these fast vertical interconnections.

An advantage of the NoC and 3D integration is that they can work flawlessly together. A NoC is made out of routers, which are connected to each other and to other CPU cores and blocks. If these routers are enhanced by vertical interconnections which enables a design of a three dimensional mesh of routers, so the NoC becomes 3D. As stated above, this in turn will enable the reduction of area size, scalability and speed improvement in the interconnection.

The drawback of this 3D design approach is that every vertical signal needs one TSV. In Section 2.1.4.1 it's explained that a TSV is a lot larger than a transistor, which means the TSVs will occupy a lot of area. The vertical signals needed can go up to a number above 160. It is essential to reduce this number of TSVs. This will be explained in chapter Chapter 4.

When making an asynchronous three dimensional design based on a 3D router, several constraints should be met. These constraints are:

- The NoC should be scalable
- It should work in 3 dimensions
- The power usage should be as low as possible
- The 3D routers should be able to operate in different clock domains
- The number of vertical lines/TSVs should be reduced

2.2.2 Components

Eventually the 3D NoC will consist of the following components:

3D router This router will be the heart of the NoC, and will have 7 ports (north, south, east, west, local, up, down). It will be able to make horizontal as vertical connections, and form a 3D connected mesh like in Figure 1.1.

Serializer The serializer serves to reduce the number of vertical signals which are going through TSVs to another stack, meaning the number of TSVs needed will decrease.

De-serializer The de-serializer serves to recover the serialized signals coming out of he TSVs in another stack, so they can be used in the new 3D router.

Synchronization Unit The synchronization unit consist of two-flop synchronizers and Gray encoders. It synchronizes the read and write pointers from the FIFO, enabling asynchronous operation of the 3D router.

Network Interface The network interface is the interface which will provide the connection between the CPU and the router through the local port.

Computational Unit The computational unit can be every type of CPU, DSP, etc. It receives data from the NoC or sends data to the NoC.

In this thesis, only a basic 3D NoC will be made out of two 3D routers, with a serializer and deserializer in between. The network interface will be left out, as well as the computational unit. Instead, this will be simulated with a injector, made in a testbench, which is able to inject flits with different injection rates and injection sizes to various network addresses. A real computational unit combined with a network interface should be able to connect to the 3D router(s), and the design should operate flawlessly.

2.3 Workflow

The first thing to do will be the design of the router. A suitable architecture should be chosen for this design which will work well with the rest of the components. Also a serializer/deserializer combination should be included for the reduction of vertical interconnections, together with synchronizers to enable the asynchronous working of the router. Second, TSV's should be designed which can be incorporated into the router design so the up and the down port can be made with it. Third, a design flow should be chosen where this design can be implemented in. For the synthesis *Synopsys Design Compiler (DC)* is chosen. The main reason for this is, that the current design flow at the faculty already uses this tool. Nonetheless it is highly optimizable and customizable for the designers needs. For placement and routing *Cadence Soc Encounter (Encounter)* is used, for the same above mentioned reasons. The simulation tool for HDL code is *Mentor Graphics ModelSim (ModelSim).* In Figure 2.3 a flow chart is given with all the steps in the design process. These steps will be described in the following paragraphs.

Step 1: Make HDL code design + testbench

First the logical behavior of the design must be specified in an HDL language. This language can be VHDL or Verilog. This logical behavior is usually described in the RTL level of abstraction, because it's most suitable for clocked designs. A complete design can consist of more sub-designs, and the usual method would be to make separate HDL

codes for each sub-design, and use these sub-designs into other designs higher in the hierarchy. To test this logical behavior of a system, a testbench must be specified. A testbench is basically also a description in HDL which supplies the top-level design with stimulus signals. A good testbench is one which tests all the relevant inputs.

Step 2a: Simulate RTL code in ModelSim

To simulate the RTL code, it must be first compiled. The order of compiling must be; first the lowest levels of abstraction, and then the higher levels. The reason for this is, that a higher level code, which uses a lower level code, cannot be compiled before the lower level code is compiled. In a simulator like ModelSim, the testbench is specified which has to be simulated, and different settings can be set. Then the simulation can be run, and the results will appear like waveforms. It can be chosen which input/output signals has to be shown, and which intermediate signals. This type of simulation only simulates the logical behavior. No delays will yet be specified.

Step 2b: See if the simulation results of the RTL code are correct

When the simulation is finished, the waveforms of the chosen signals can be studied. Here it can be observed if the logical behavior of the RTL code is correct. If there are signals that are false, or even worse, unknown, the RTL code has to be changed. And the simulation has to be ran again.

Step 3: Set DC setup constraints

Now that the RTL code is logically valid, it should be synthesized into design which exist of only the standard cells and macro's from the different libraries. For this a tool is needed like Design Compiler. Design Compiler can be ran in graphical mode, and in batch mode. Batch mode is more efficient, especially because scripts can be used to run the tool. These scripts are written in the TCL language. In these scripts different needed files have to be specified. Also various constraints have to be specified, like clock speed, clock latency, input/output delay, etc. These constraints will determine the final synthesized design.

Step 4a: Synthesize in Design Compiler

After all the scrips have been finished, they can be executed, and all the commands with their respectively constraints will be executed. In turn the Design Compiler will map the RTL design on the standard cells and macro's from the available library.

Step 4b: Look at the output reports from Design Compiler and see if it meets the constraints

Different output reports will be generated by Design Compiler, ranging from the number of gates used, to the amount of area used, to different timing reports that are the most important reports. The timing reports specify the paths between two clocked devices (registers and/or memories) or between the input/output en other clocked devices,

with the largest negative slack. In the scripts, special commands can be given which paths should be put into the timing reports. This is for debug purposes. If the timing is correct, it should be simulated for correction. If not, the constraints should be adjusted or the RTL code should be changed.

Step 5a: Simulate synthesized code in Modelsim

This is basically the same step as step 2a, only here the synthesized code is simulated together with a *SDF (standard delay format)* file. The synthesized code is a structural description of the design, with all the standard cells and macro's (memory blocks) connected to each other in the best possible way, for a functional and fastest design. The SDF file contains of delay values for the used standard cells, and the interconnect delays are guessed, because Design Compiler doesn't contain any interconnect information. These delay values are annotated into the synthesized design, and it is simulated.

Step 5b: See if the simulation results of the synthesized code are correct

As in step 2b, the waveforms can be seen, only this time with the delay information available. Again, if there are signals that are false, of unknown, something can be wrong with the simulation settings or synthesis settings.

Step 6: Set SoC Encounter setup constraints

Now that the synthesized code is valid, it should be placed and routed. Here the real interconnect delay values will be calculated and extracted. For this a tool like SoC Encounter is needed. This tool can also be run in graphical mode, and in batch mode. Also here scripts can be used which are made in the TCL language. Here different commands and constraints are specified which have to be executed. These constraints will determine the final placed and routed design.

Step 7a: Perform place&route in SoC Encounter

Apart of the constraints that have to be specified in the scripts, and various TCL commands, there are some things that have to be done manually. One of these things is the placing of the macro blocks and the inclusion of power rings, lines and stripes. This has to be done in the graphical way. A good thing about it is, that when these things are done the first time, the TCL commands used by encounter to do the placing and the power design can be found and used in a script for future execution. After the execution of the scripts, and eventual manual adjustment of the macro placement and inclusion of the power, SoC Encounter will place all the standard cells and macros into a real die, and interconnect them with signal lines and power lines.

Step 7b: Look at the output reports from SoC Encounter and see if it meets the constraints

SoC Encounter gives output reports with various information like Design Compiler does in step 4b. The timing reports here are also the most important, because it shows if

the design will work at the specified clock speed. If all these reports are satisfactory, it should be simulated for correction. If not, some design constraints or placement has to be changed to meet the specified constraints.

Step 8a: Simulate after place&route the final code in ModelSim

Here the simulation of the placed and routed design is done, together with the SDF file. Only this time the SDF file contains real interconnection delays, so this simulation should be the most comprehensive.

Step 8b: See if the simulation results of the final code are correct

These results are of a 'real' design, which is placed and routed, and where real delay values are used for simulation. If these simulation results are correct, this design is valid, and can be taped out or used as a macro block for another design.

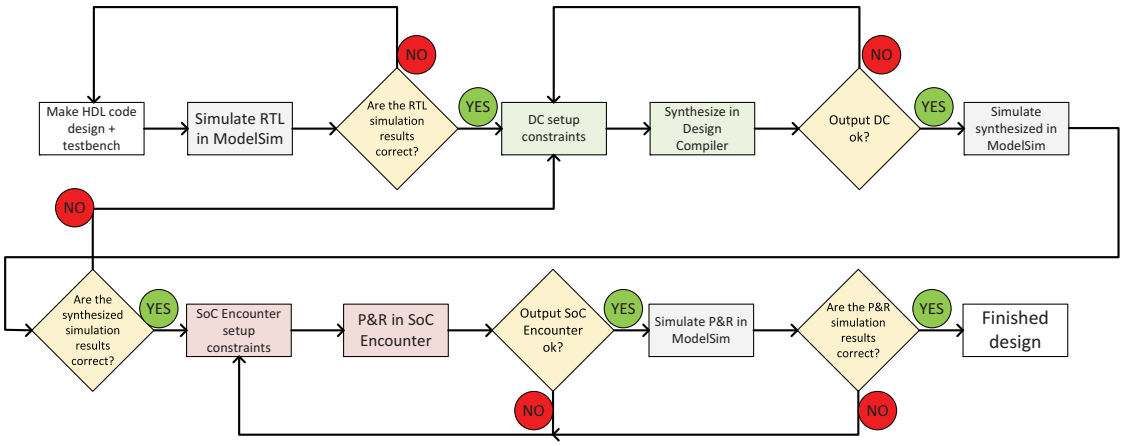


Figure 2.3: Flow chart of the design steps

3D Router

3

This chapter describes the basic design of the 3D router, which is the main building block of a 3D NoC. This excludes the serializer/deserializer and synchronizers, which will be discussed in chapter Chapter 4 and Chapter 5. First a short motivation will be given on the design methodology for the 3D router in Section 3.1. The architecture of the 3D router is described in Section 3.2, followed by the description of the memory and the through silicon via in Section 3.3 and Section 3.4. In Section 3.5 a detailed physical implementation is given of the basic 3D router design.

3.1 Motivation

The 3D router is the heart of the 3D NoC. It routes all the signals from source to destination, so all the attached computational units are able to communicate with each other in an efficient way. When designing this 3D router, it is very important that it is suitable for the complete system, which includes the complete 3D NoC and the attached computational units.

This chapter is about the design of the 3D router, using existing technology and tools. The emphasis will be on the implementation of the TSVs, because this is a novelty in the design process. This implementation of the TSVs will not be conventional, in the sense that the end result won't be able to be taped out as a physical chip. The TSV implementation will be based on an abstract representation, which will suffice for synthesis, placement&routing. For real implementation of the TSV, this physical representation can be changed into a real one, but the timing shouldn't change much.

3.2 Architecture

The first step of the router design is to chose a suitable architecture. This architecture should be able to support the specific function of this router, which means it should also be able to work in 3D. The architecture of choice is the input buffered wormhole router which is designed by S. Kumar [6]. This architecture was built initially for a 2D 5-port router with the possibility to extend it to a 3D 7-port router. With small adjustments in the RTL code, the two extra ports (UP and DOWN) were added so a fully functional 3D router was obtained. In this chapter the 7-port router will be discussed, which means the two extra ports will function as vertical interconnects for up and down communication and will be implemented with TSVs. This router is developed in-house at TU Delft as a part of the TMFab project by S. Kumar, so this router is easily obtainable and many documents are available for extra information needed for it's ASIC implementation. The architecture is shown in Figure 3.1, where

all the different components are pointed out, and the input and output signals are shown.

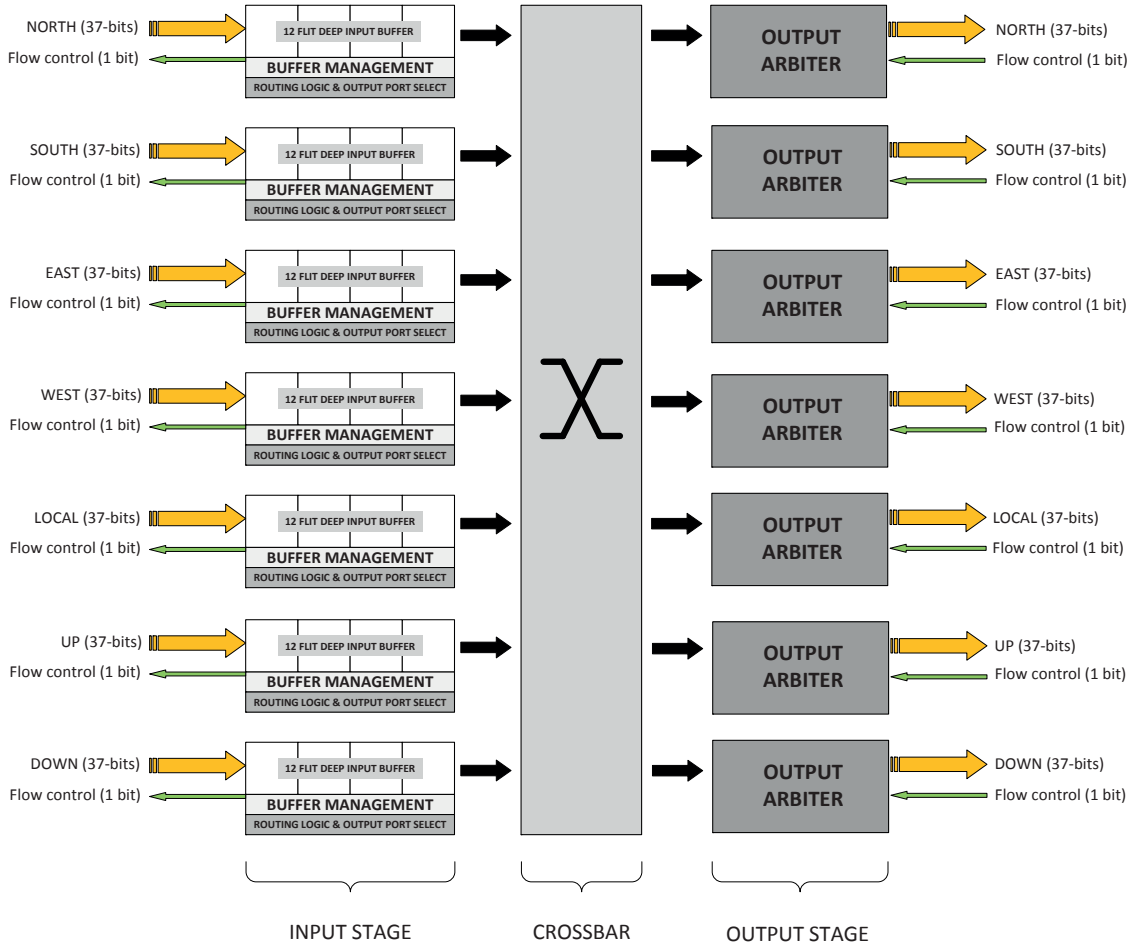


Figure 3.1: Three dimensional 7-port router architecture

The input stage consists of the 7 input ports (each port for one direction), where each input port consists of a input buffer part, buffer management part, and the routing logic&output port select part. The output stage consists of 7 output arbiters (each arbiter for one direction). Between the input stage and output stage, there is a ordinary crossbar which links the input ports with the output arbiters. At each input port, there is a 37-bit wide input signal, and a 1-bit output signal. At each output arbiter, there is a 37-bit wide output signal, and a 1-bit input signal. The 37-bit signals are data bits, and the 1-bit signals are the flow control bits.

The used architecture uses *wormhole routing* for routing data packets across the network. These data packets are divided into flits. Each flit consist of 37 bits. That is the reason the complete NoC supports 37-bit input and output links interconnecting nodes. Each flit consists of a *header* flit, a *body* flit and a *tail* flit. The main task of

the header flit is to carry the *destination address* for the complete packet. The body flit contains the payload data, just like the tail flit, where the tail flit is the last one, and marks the end of the packet. This division of flit types enables the data to be send flit by flit, without waiting for each other at each node. This means first the header flit is routed across the different routers, where all the other flits simply follow the header. Because there is no waiting for other flits at each node, this improves the routing time significantly. The different flits are specified in Figure 3.2.

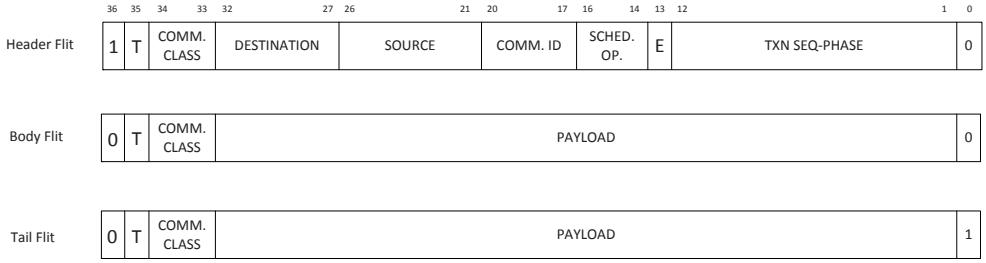


Figure 3.2: Packet Format

Each tile has a unique *Local Network Address*, where a tile consists of a router and a computational unit which is connected to the local port of the router using a network interface. This address serves to identify each tile. When a header flit arrives at a router, and it's destination address matches the local network address of the tile, this flit together with the other flits are transferred to this particular router and its computational unit, where further processing is performed.

Network packets may contain up to 64 bytes of payload data. This payload data is transferred as 37-bit flits, where each flit contains 4 bytes of the payload data. This means a network packet can be at most 17 flits long (16 flits for the payload and 1 header flit).

The routing algorithm used in this three dimensional router is based on Z-X-Y routing, which means the data is first routed in the Z direction, and subsequently in the X and Y direction. This ensures that the packets injected into the network are transferred directly to their destination layer through the TSVs, which will reduce the congestion in the layer meshes. In the end, this algorithm prevents the incoming packets from being routed through the same port as where they were received at.

In this architecture each packet is routed independently through the network, so it eliminates the need for a connection set up. If a packet is injected into the network, eventually it will reach its destination flit by flit. There are also situations when a link at a router is busy, so a flit has to wait at that router. And when that link is free, it can be processed to the next router. In the mean time this flit has to be put in some kind of buffer. The buffer has a certain pre-specified size, which means only a limited number of flits can be stored there and eventually it can fill up. When this happens, the occupation of the input buffer should get under a certain threshold before processing next flits through this particular input port. For this purpose there is a flow control mechanism which ensures that flits are never dropped, thereby eliminating the need for retransmission. This is implemented by a flow control bit, which gets high when the occupation of the downstream input port buffer gets to a specified threshold value.

This value is computed by N-2. In this particular router the value of N is 12. N is the number of input buffer slots each 37-bits wide, which means in every slot, 1 flit can be put. The value of N is determined from simulations by examining the variation in average end-to-end latency and throughput with increasing input buffer depth [6]. In Section 6.1.2 the FIFO depth will be changed to test it's influence on the throughput of two routers connected to each other. The pipelined nature of the router induces a two-cycle latency for transmission stalls in the event of congestion at a downstream router.

The output arbiters use a *Round Robin* arbitration scheme, which ensures fair and best effort service to all input ports. Because the routing algorithm prevents the packets being routed the same port they arrived at, each output arbiter will poll only 6 of the inputs. When a header flit arrives at the input port, the Routing Logic & Output Port Select part raises a request to the appropriate output arbiter. If the requested output port is idle, and the downstream input port has free input buffers left, the arbiter grants the request by asserting the Drain signal for the input buffer. After this, the round robin arbitration is deactivated until the tail flit is passed through the output port. Hereby the integration of the routed packets is ensured, and it prevents the flits from other packets mixing in the output stream.

Subsequently the input port waits for access to the output port, which means keeping its request lines asserted and input buffers stalled. If the input buffers become empty while flits are advancing through the router, the output arbiter sets the link as idle while maintaining the round robin arbitration in the deactivated state. This state holds until the tail flit has advanced and the complete packet has been routed. After completion, the arbiter asserts the Next signal and resumes round robin arbitration to service the next waiting input stream.

This type of router has a minimum fall through latency of four cycles.

3.3 Memory

First the input buffer should be implemented by a real memory block. For this the 55 nm synchronous, two port register file from Faraday is chosen, as stated in Section 2.1.3. The specifications are given in Table 3.1. In the router design, it is determined that each input buffer should be at least 12 flits deep and 37 bits wide [6]. This means the memory should be 12 words deep and 37 bits wide. The number of words is determined by the address length, as seen in (3.1), where W is the number of words and A the address length.

$$W = 2^A \tag{3.1}$$

Because the number of words can only be a power of two, a value of 16 is chosen, where 4 words will stay unused. The word width is exactly the same as specified in the architecture. For testing purposes, memory blocks with different word depths will be used in Chapter 6.

The problem with this memory block is that it is made in the 55 nm technology instead of the 45 nm technology used by the Nangate standard cells. Some adjustments should be made in the LEF file, which describes the physical properties of the block in

Table 3.1: Memory block specifications

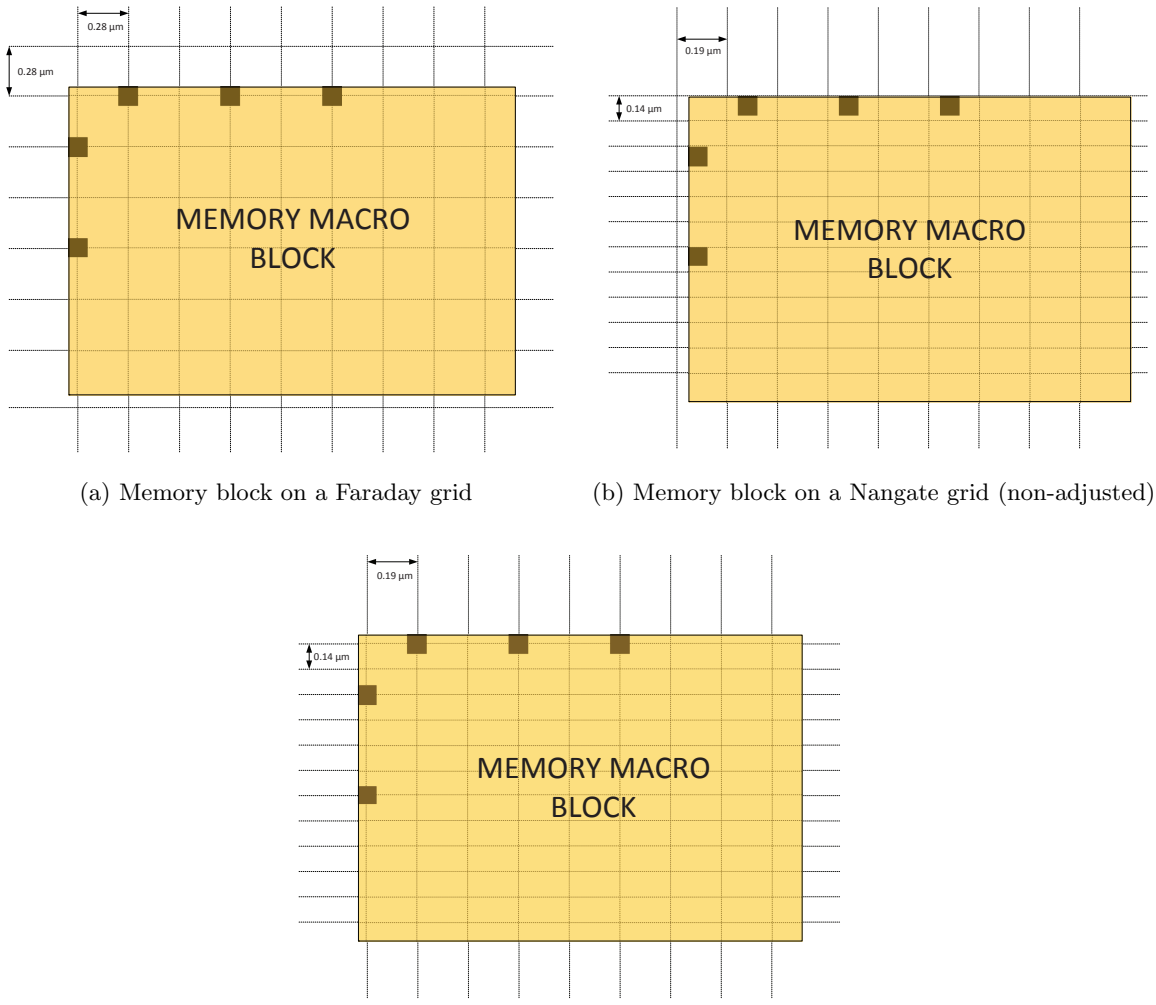
Physical length	226.8 μm
Physical width	36 μm
Number of words	16
Bits per word	37
Address length	4
Write cycle time (WC)	1.25 ns
Write cycle time (BC)	0.49 ns
Read cycle time (WC)	1.25 ns
Read cycle time (BC)	0.49 ns

an abstract way. It is important to explain that all the cells, blocks and routing layers sit on an grid. Every technology has it's own grid, which means the Nangate library and Faraday library use different grids. The Faraday routing grid is 0.28 micron in both the X and Y direction, where the Nangate routing grid is 0.19 micron in the X direction and 0.14 micron in the Y direction. Because this is an ASIC semi-custom design, it means only abstract descriptions are used for the different blocks and cells. In this particular case it means that the coordinates of the memory pins should be adjusted in such a way that they align with the same grid as the Nangate standard cells. After this is done, all the blocks and cells including the memory can be routed. This doesn't mean that it's sufficient for fabrication, because the transistors in the memory block are still in 55 nm technology. This adjustment is only meant to enable the routing of the memory blocks in the place&route tool. In Figure 3.3 three memory blocks are given on a specified grid. Figure 3.3a describes the original situation where the memory block sits on a Faraday routing grid, which means all the pins are on the grid. Figure 3.3b describes the non-adjusted memory block sitting in a Nangate routing grid, where the pins are not sitting on the grid. Figure 3.3c shows the adjusted memory block for the Nangate routing grid where the pins are perfectly snapped on the grid. The adjustment of the memory pin coordinates is done by hand.

3.4 Through Silicon Via

The TSV is the component which enables the vertical ports of the router. In Section 2.1.4 some background information is given about the technology needed for TSV design. At the end of that section, the available TSV technology from the NCSU FreePDK library is discussed. This section will focus on the implementation of the TSV using the findings of R. Jagtap [8]. Because the TSV dimensions from the NCSU FreePDK library are comparable with the TSV dimensions in [8], and a lot of information is given about the timing and placement of TSVs, these findings will be used for the implementation and calculation of TSVs in this thesis.

First the implementation of this technology into the design flow will be discussed, where the TSV should be designed in such a way, that it can be implemented by the tools into the design. Subsequently the placement of the TSVs will be discussed in the



(a) Memory block on a Faraday grid

(b) Memory block on a Nangate grid (non-adjusted)

(c) Memory block on a Nangate grid (adjusted)

Figure 3.3: Memory block grid adjustment

router design.

3.4.1 TSV Design

First the TSV has to be designed, which later can be incorporated into the router design. In Section 2.1.4.4 the technology from the NCSU FreePDK3D45 PDK is explained, and together with the findings from [8] they form a good foundation which the TSV design will be based on. The NCSU FreePDK3D45 PDK is a design kit compiler for stacked dies (i.e. 3D-ICs) in a predictive 45 nm technology [24]. This kit contains the basics of what is needed to perform schematic entry, SPICE simulation, layout, DRC, and LVS checks. The design flow assumes a stackup like in [1]. Because in this design the most important aspect is the timing and floorplan analysis, the TSV can be made in a more abstract way, so all these files from the PDK are not really needed. The

only thing of importance here is that the TSV area specifications are comparable with the PDK. So when a future designer wants to use all the PDK files, the synthesis and place&route step can be done without large modifications. It is chosen to design the TSV in an abstract way, and to incorporate the timing information from R. Jagtap [8]. In this design process, first the specifications of the TSV need to be pointed out. The specification which is important for this design is basically the diameter of the TSV. The diameter is 4 μm, which is the used diameter for the TSVs of R. Jagtap. This means that this will be the constraint that will be used in the TSV design. The files that are needed for the design are listed as follows:

- An HDL description, which is needed by the synthesis for determining the logical functionality and connection
- LEF file, which is needed by the place&route for determining its physical form and layout
- Timing information, which is needed for timing simulation

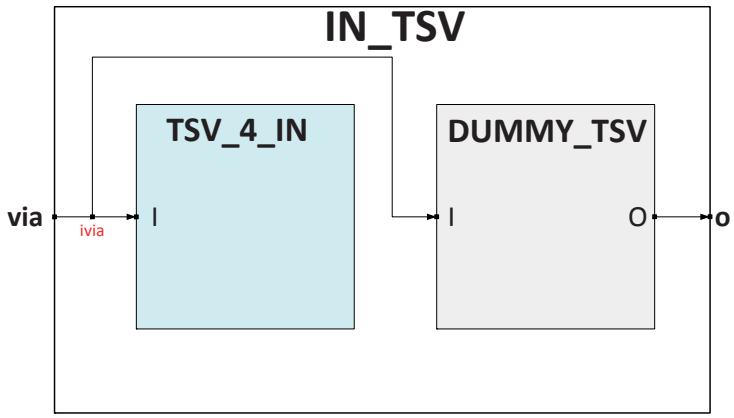
3.4.1.1 TSV HDL Description

For a TSV to be used, it should somehow be incorporated into the complete router design. This means that it should be modeled in a HDL. Because a TSV is not an ordinary component with an input and output port, a special construction has to be made which describes the behavior of the TSV. A current component which comes to mind with similar properties, is the chip pad. A chip pad is also a stack of metals, which can be used as an input or output to the outside world. In this sense, the TSV is the same thing, only in this case the outside world is the other stack (UP or DOWN) to which this TSV will be connected to.

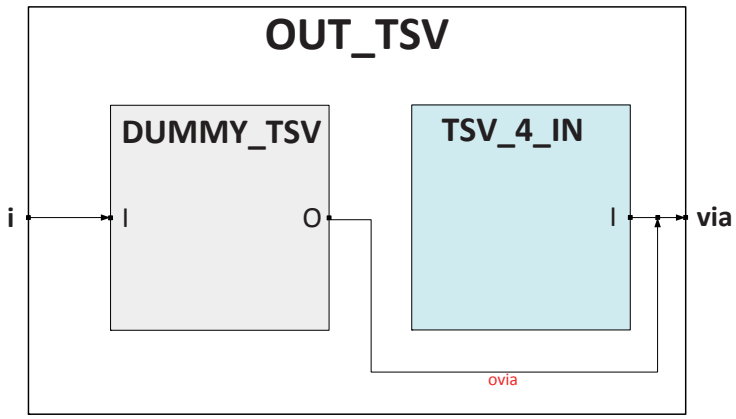
This HDL description is needed for simulation purposes, and for synthesis and place&route purposes. For simulation purposes it is necessary that the TSV is connected to the vertical input and output ports of the router. This means that a vertical output signal from the router should go into a TSV, which in turn means the TSV should have an input port. But at simulation, this output port should be read out from the TSV, so the TSV should have also an output port.

A tried construction for the HDL description is shown in Figure 3.4, where in Figure 3.4a the construction for the HDL description of the input TSV is shown and in Figure 3.4b the construction for the HDL description of the output TSV is shown. Here it can be seen that a TSV is modeled out of two other components. This is needed because the TSV doesn't have an ordinary input and output port. This way the place&route tool had to be tricked not to route the TSV input/output port to the side of the router block, because it doesn't know the connection is vertical. After place&route the TSV had no input/output port at all in the structured HDL description, which meant it couldn't get connected to another 3D router. A solution for this problem was to use a normal construction with a ordinary input and output port for the TSV, which is shown in Figure 3.5. It was calculated that the delay in the connection between the TSV and the side of the router block (which didn't exist in reality) was

between 2 and 23 ps. Because the longest path in the network was in the order 4 ns, this had no significant influence on the timing.



(a) HDL design for input TSV



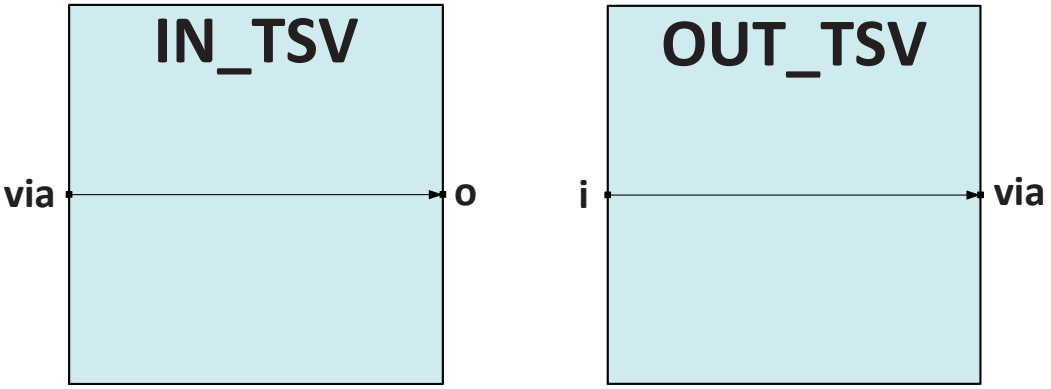
(b) HDL design for output TSV

Figure 3.4: Tried HDL construction for TSVs

3.4.1.2 TSV LEF File

Because this is an ASIC semi-custom design, for every standard cell or macro cell a *Library Exchange Format (LEF)* file is needed. A LEF file gives the physical description of such a cell [25]. In this case a TSV will be implemented. This means a LEF file is needed to describe the physical properties of this cell. Because a TSV is not a standard cell, it will be implemented as a macro cell with it's own physical properties.

A LEF file consists usually of two parts, one technology part and one macro part. The technology part contains the technology information of the design. This includes placement and routing design rules and process information for the layers. The macro part contains the standard cell and macro definitions for a design. These two parts can be put into one LEF file, or in two separate LEF files. One technology LEF file and one macro LEF file. Even the macro part can be separated into different LEF files



(a) HDL design for input TSV (b) HDL design for output TSV

Figure 3.5: Used HDL construction for TSVs

for different cells. Because there is already a technology file and a macro file for the Nangate standard cells and the Faraday memory, an extra LEF file will be made which describes the TSV macro cell. The macro definition defines the macro, it's size and it's pin definitions and physical characteristics of each of the macro pins.

A macro definition begins with a macro name followed by the type of macro which is called a CLASS. The **BLOCK** class is chosen. The reason for this choice was that in this case the place&route tool sees this as a ordinary macro. The only problem is that the TSV will have an input and output port, where the output port is routed by the place&route tool to the side of the router block, which doesn't exist on a real chip. But because of pragmatic reasons, this is left this way. After this a FOREIGN specification is given, which specifies the GDSII structure name that is to be used, and where the coordinates specify the macro origin as offset from the foreign origin. The name chosen is the same as the macro name, and both the x and y coordinates as 0.000. The physical model of this TSV is described as an black-box in an ASCII form. This abstraction is needed because during the place&route stage, this model will be used for the placement and routing. A small amount of information is needed for this task, and because of the abstraction, the file becomes very small, which increases the speed of the place&route stage. This blackbox is defined as an rectangular area. First the x and y coordinates are given for the origin of this macro cell, where the rest of the coordinates are specified with respect to this origin. Usually the coordinates are chosen to be 0.000, which is also the case for the TSV. The size specification defines the minimum bounding rectangle of the macro, which includes all pins and blockages. Subsequently some symmetry statements are given to define how the place&route tool can mirror or rotate the cells in the layout of the design. Because this is just a TSV, it can be rotated and mirrored in every possible way. Every macro description has its own SITE specification. This defines a placement site in the design. It is important to be compatible with the technology LEF file and the site defined therein, which is **FreePDK45_38x28_10R_NP_162NW_34O**.

All the pins of the macro have to be defined. However, these are not real pins in the design, but predefined areas where the routing tool knows to route the metal

interconnections. First the name of the PIN has to be specified, which has to be the same as the pin name from the HDL description. There are 2 pins, from which **I** is the input pin and **O** is the output pin. A pin specification consists of port statements, which defines a set of geometries that are electrically equivalent points. Each port has a layer statement, which defines the geometry of a metal layer. With the RECT statement, the geometries are given with 4 coordinate points. The first two are the x and y coordinates of the lower left corner, and the second two are the coordinates of the upper right corner.

At the end of each macro description there is a OBS statement that consists of LAYER and RECT statement just as in the PIN specification. The only difference is that these areas are obstruction areas, which means no metal wire can be routed through these areas. Because a TSV occupies all the metal layers and goes through the silicon, it is impossible for a metal wire to be routed through the TSV. This is modeled by specifying the obstruction area as the whole TSV area.

Because only rectangles can be specified in a LEF file, it is decided to implement the TSV as a rectangle and not as a circle. The TSV layout which is defined in the LEF file is shown in Figure 3.6. The dimensions of the TSV are specified by A and B, which are both 4,06 μ. The reason that the dimensions are 4,06 and not 4 μ, is that it is a multiple of the horizontal grid in the Nangate technology. This ensures the TSV is always on the grid to be connected. The ports are specified by rectangles 1 to 4, where rectangles 1 and 4 have the dimensions A by C, and rectangles 2 and 3 have the dimensions D by E. C and D are 1 μm and E is 2,06 μ. In this way an outer ring is formed with width D. Rectangles 1 and 3 are specified as input ports, where rectangles 2 and 4 are specified as output ports. The given TSV sizes can be always changed, for other TSV dimensions.

3.4.1.3 TSV Timing

A TSV is a slab of metal which goes through silicon, and which connects two stacks with each other. It has it's own parasitic effects which define the timing of the interconnect. It's placement will define what these parasitic effects will be. A TSV has a different behavior when sitting by it self in an area, than when it is in an array of different TSV's, which will be the case in this design. The timing is defined by the *Capacitance (C)*, *Resistance (R)* and *Inductance (L)*. The values have to be found for these factors. R. Jagtap states in [8] that the closed form equations from [26] are sufficient for the description of the three electrical factors. All the factors are calculated to work at a high frequency. The reason for this is, that the complete router design should work at a frequency as high as possible. The equations are given in the following paragraphs. The equation parameters are given in Table 3.2.

Resistance of TSV The TSV resistance R_{HF} (Ω) is given in (3.2)

$$R_{HF} = \begin{cases} \alpha \frac{\rho_m L}{\pi[R^2-(R-\delta)^2)]} & \text{if } \delta < R \\ \alpha \frac{\rho_m L}{\pi R^2} & \text{if } \delta \geq R \end{cases} \tag{3.2}$$

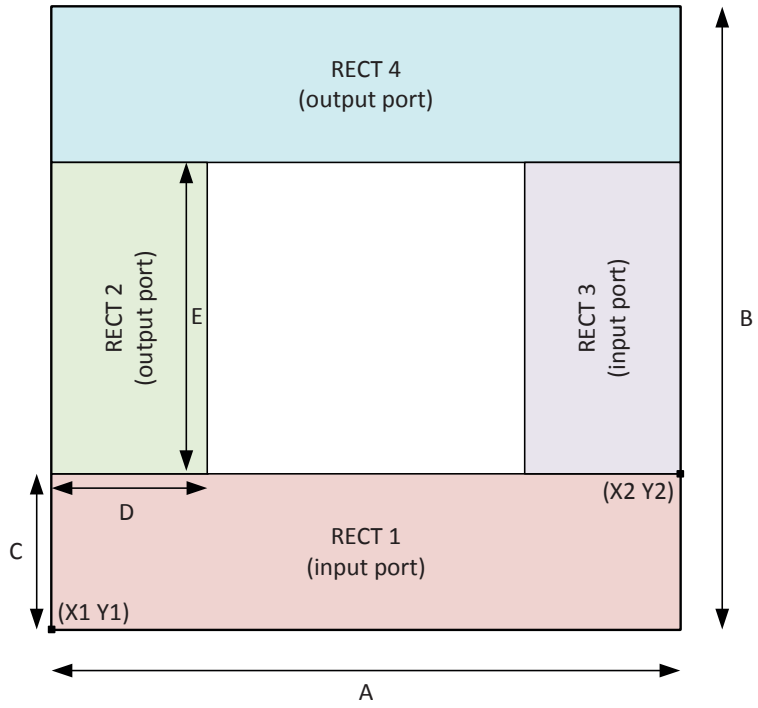


Figure 3.6: Layout of a TSV from a LEF macro definition

The skin depth δ (m) is given by (3.3)

$$\delta = \frac{1}{\sqrt{\pi(1/\tau)\mu_0(1/\rho_m)}} \tag{3.3}$$

and the α factor is given by (3.4), which describes the fitting parameters

$$\alpha = \begin{cases} 0.0472 D_{\mu m}^{0.2831} \ln(\frac{L}{D}) + 2.4712 D_{\mu m}^{-0.269} & \text{if } \delta < R \\ 0.0091 D_{\mu m}^{1.0806} \ln(\frac{L}{D}) + 1.0518 D_{\mu m}^{-0.092} & \text{if } \delta \geq R \end{cases} \tag{3.4}$$

Inductance of TSV The TSV inductance is divided in L_{11} (H) for partial self inductance, and in L_{21} (H) for mutual inductance. These equations are given by (3.5)

$$HF : \begin{cases} L_{11} = \alpha \frac{\mu_0}{2\pi} |\ln(\frac{2L}{R}) - 1| L \\ L_{21} = \beta \frac{\mu_0}{2\pi} [\ln(\frac{L+\sqrt{L^2+P^2}}{P}) L + P - \sqrt{L^2+P^2}] \end{cases} \tag{3.5}$$

where α and β are given by (3.6) and (3.7)

$$\alpha = 0.94 + 0.52 e^{-10|\frac{L}{D} - 1|} \tag{3.6}$$

$$\beta = 0.1535 \ln\left(\frac{L}{D}\right) + 0.592 \tag{3.7}$$

Capacitance of TSV The TSV capacitance is given by (3.8)

$$C = \alpha\beta \frac{\epsilon_{diel}}{t_{diel} + \frac{\epsilon_{diel}}{\epsilon_{sub}} x_{dTp}} 2\pi RL \tag{3.8}$$

where α and β are given in (3.9) and (3.10)

$$\alpha = \left(-0.0351\frac{L}{D} + 1.5701\right) {S_{gnd_{\mu m}}}^{0.0111\frac{L}{D} - 0.1997} \tag{3.9}$$

$$\beta = 5.8934 {D_{\mu m}}^{-0.553} \left(\frac{L}{D}\right)^{-(0.0031 D_{\mu m} + 0.43)} \tag{3.10}$$

and the depletion region depth in p-type silicon is given by (3.11)

$$x_{dTp} = \sqrt{\frac{4\epsilon_{sub}\phi_{f_p}}{qN_A}} \tag{3.11}$$

$$\phi_{f_p} = V_{th} \ln\left(\frac{N_A}{n_i}\right) \tag{3.12}$$

The coupling capacitance between two TSV's is given by (3.13)

$$C_c = 0.4\alpha\beta\gamma \frac{\epsilon_{sub}}{S} \pi DL \tag{3.13}$$

where α, β and γ are given in (3.14) to (3.16)

$$\alpha = 0.225 \ln\left(0.97\frac{L}{D}\right) + 0.53 \tag{3.14}$$

$$\beta = 0.5711 \left(\frac{L}{D}\right)^{-0.988} \ln(S_{gnd_{\mu m}}) + \left(0.85 - e^{-\frac{L}{D} + 1.3}\right) \tag{3.15}$$

$$\gamma = \begin{cases} 1 & \text{if } \frac{S}{D} \leq 1 \\ \zeta \left[\ln(\frac{L}{D} + 4e^{-\frac{S_{\mu m}}{9}} + 2.9) - 10.625 {S_{\mu m}}^{-0.51}\right] & \text{if } \frac{S}{D} > 1 \end{cases} \tag{3.16}$$

and ζ in (3.17)

$$\zeta = \left(1 + e^{[(0.5 + |\frac{P}{D} - 4|)\frac{L}{D}]}\right) \tag{3.17}$$

The TSVs in this thesis are of the same size as the TSVs used in [8]. This means that the same values for C, R and L can be used. The only difference being that the LEF model in this thesis uses square terminals, which is the only possible abstraction in a LEF description. This doesn't have influence on the timing of the TSVs, because for timing simulations the TSV is modeled as a circle. The C, R and L values for the TSV are already determined in [8], which means these can also be used for the TSVs in this thesis. These values are shown in Table 3.3. The delay is determined mostly by the C and R values. If a wire has a minimum length of 20 micron which connects a TSV to a

Table 3.2: Equation parameters with their symbols

Symbol	Parameter
D	TSV diameter (m)
R	TSV radius (m)
P	TSV pitch (m)
L	TSV length (m)
S	Spacing between two TSVs (m)
t_{diel}	TSV thickness of dielectric layer (m)
S_{gnd}	Distance of TSV from ground plane (m)
ρ_m	Resistivity of TSV fill metal $\Omega.m$
N_A	Doping concentration of p-type Si substrate $(/m^3)$
ϵ_{sub}	Permittivity of Si substrate (F/m)
ϵ_{diel}	Permittivity of dielectric layer (F/m)
μ_0	Permeability of free space (H/m)
τ	Clock edge (s)

component (this is because of the keep-out-zone which lies 20 micron around the TSV array), and the C, R values for a 45 nm technology are given to be 0.20 $fF/micron$ and 9.46 $\Omega/micron$ [8], the C and R values for that wire are 4 fF and 189.2 Ω. Compared to the values from Table 3.3, the C is a factor of 5 smaller, where the R is a factor of 700 larger. In the end, the RC delay for the TSV is much smaller compared to the wire delay in the 45 nm technology.

Table 3.3: Values for the parasitic components of a TSV

Component	**Value**
R_{TSV}	0.24 Ω
L_{TSV}	7.50 pH
C_{TSV}	21.60 fF
C_C	2.75 fF

To be able to make a Liberty file, where propagation delay and output transition values must be given w.r.t. input transition time and output capacitance, a schematic has to be made where this can be simulated. The same schematic can be used as in [8], where two capacitive coupled TSV's with their own parasitic values are simulated. This is shown in Figure 3.7, which is simulated in a Spice software called Cadence OrCAD. In this simulation, both of the voltage sources are switched on simultaneously. The same values for the input transition time and output capacitance are used as for the Nangate liberty file. All the results are given in Appendix A, where it can be seen that the output transition time is almost equal to the input transition time, and the propagation delay is negligible compared to the propagation delay in the Nangate technology. For the liberty file the output transitions are held equal to the input transitions and the propagation delay is set to zero.

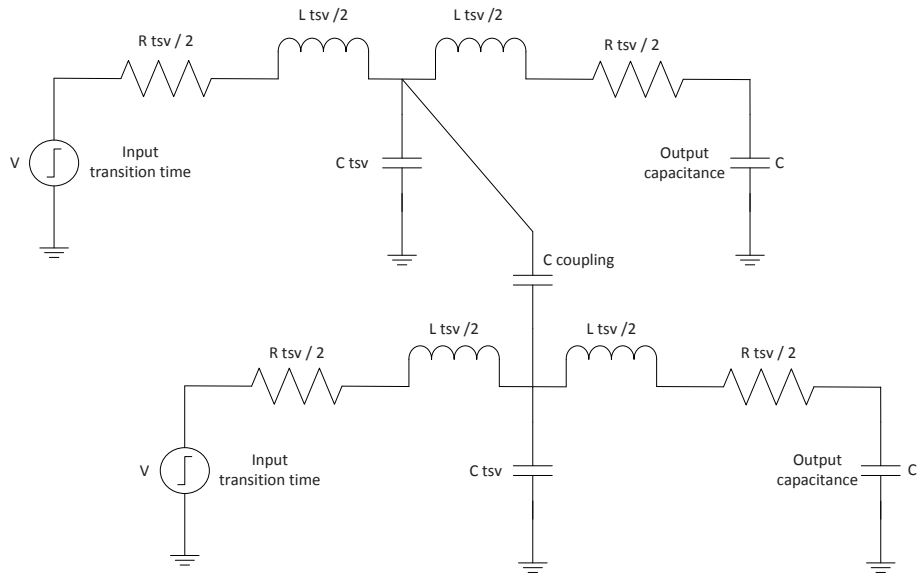


Figure 3.7: Circuit for simulating delay and output transition time w.r.t. input transition time and output capacitance with the influence of capacitive coupling

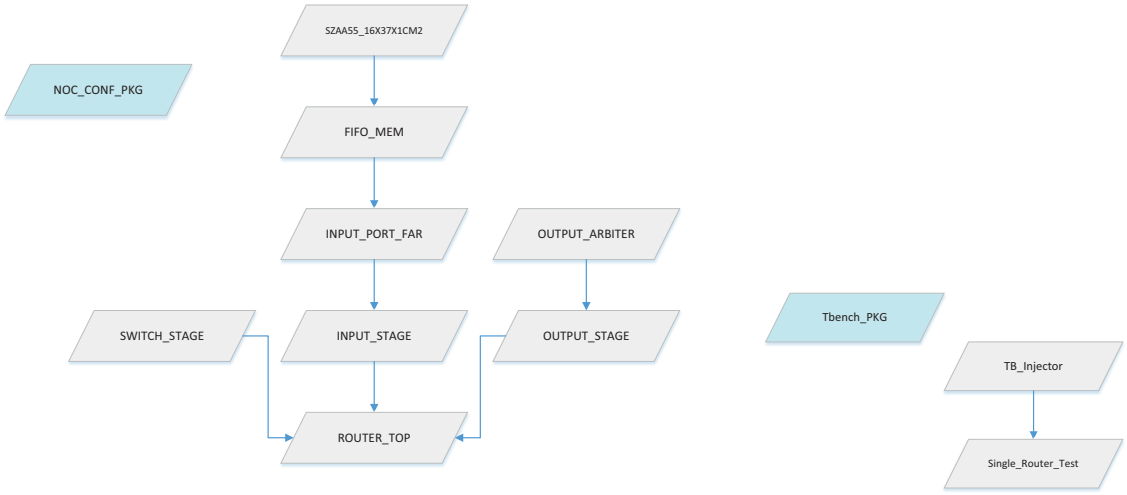
3.5 Implementation

3.5.1 Implement the RTL HDL code and testbench

The first step is to gather the RTL code, which is written in different VHDL files and structure it. When having a structure, it is easier to understand the code when needed for debugging or adjusting. These files are obtained from S. Kumar [6] as stated in Section 3.2. To test this code, a testbench is made, which also consists of different VHDL files. Figure 3.8 shows the complete structure of these files. In Figure 3.8a the structure of the RTL code is shown, where in Figure 3.8b the structure of the testbench is shown.

3.5.2 Simulation of RTL

The simulation that is performed is based on a testbench that tests the local input port, and all the other output ports. Basically there is a injector component that serves as a computational unit which is attached to the local port of the router. This injector injects flits into the local port of the router, and these flits have to be routed to the output ports of this router. When the buffer of the local input port arrives at threshold value, the injection stops, and when the input buffer gets under the threshold value, the injection continues. In this way, the complete routing function is tested. Because the simulation of the RTL code is just a logical simulation and no timing information is available, it is the best-case scenario, so it basically determines the maximum throughput of the router. The maximum throughput will not be determined for this basic 3D router, because in chapter Chapter 4 and Chapter 5 more functionality



(a) RTL file structure

(b) Testbench file structure

Figure 3.8: VHDL file structure

will be added. In chapter Chapter 6 the throughput will be determined of two vertically connected 3D routers with a serializer/deserializer in between, where the injector, and the routers work all on a different clock frequency. This way, the most important connection in a 3D NoC will be tested.

3.5.3 Synthesis in Design Compiler

Now that the RTL HDL code is validated, it can be synthesized into standard cells. For this purpose a synthesis tool is needed, which will be Synopsys Design Compiler. First the data has to be prepared, and then the flow should be specified to carry out the synthesis process. The files needed for this process are:

.synopsys_dc.setup This is the setup file for Synopsys Design Compiler. In this file paths and file names are specified for the liberty files of the used standard cells and macros.

NangateOpenCellLibrary_slow.db This is a liberty file in a binary format which can be used by DC. In this file timing information is given for all the Nangate standard cells working in slow (worst-case) conditions. The used delay format is *NLDM (Non Linear Delay Model).*

NangateOpenCellLibrary_fast.db This is a liberty file in a binary format which can be used by DC. In this file timing information is given for all the Nangate standard cells working in fast (best-case) conditions. The used delay format is *NLDM (Non Linear Delay Model).*

SZAA55_16X37X1CM2_WC.db This is a liberty file in a binary format which can be used by DC. In this file timing information is given for the Faraday macro

working in WC (worst-case) conditions. The used delay format is *NLDM (Non Linear Delay Model)*.

SZAA55_16X37X1CM2_BC.db This is a liberty file in a binary format which can be used by DC. In this file timing information is given for the Faraday macro working in BC (best-case) conditions. The used delay format is *NLDM (Non Linear Delay Model)*.

mblite_syn_7.tcl This is a script file where all the commandos are given to run DC, together with all the design constraints.

8 RTL VHDL files This are the VHDL files which together describe the complete router design. The logical behaviour from these files will be used, and mapped into the specified technology.

For the synthesis of this design, two different libraries from two different vendors will be used as stated in Section 2.1. The reason for this is given in Section 2.1.1. The 45 nm Nangate library will be used for the synthesis of the standard cells, where the 55 nm Faraday library will be used for the synthesis of the memory modules. It is important that the synthesis tool can read these two libraries without a problem, and use their timing information for synthesizing the best possible design. The 45 nm Nangate library is the main library, which means the operating conditions set in the Faraday library should be adjusted, so they match the value of the Nangate library. In this case it means the original value of **BCCOM** should be changed to **fast**. In the script file, where the operating conditions are specified, these should be set to **slow**, and the wire load model to **5K_hvratio_1_1**. These are the worst-case conditions from the Nangate library. The reason why the worst-case conditions are given is that if a design operates under these conditions, the probability of operating under all other or better conditions is large. The only problem is when the design is too fast, hold violations will occur. This means that a signal will arrive too soon at a flip-flop, in the time the previous signal isn't properly clocked in yet. The chances of these violations to occur are far smaller compared to the chance that the signal is too slow.

Different clock periods are used to synthesize the design. No clock latency values are specified in this design, because DC doesn't have accurate timing information about the wire delay. The place&route tool will determine this value. Only the driving cell for the clock will be specified because for simulation purposes it is not practical to use an ideal clock with no clock rising or falling time. A simple cell with standard strength will be used to model the non-ideal clock input. This cell is a simple inverter with 8X the strength **INV_X8**. This same cell is also used as the input to all the input ports of the 3D router design, because in real-life this input is not ideal.

3.5.4 Place and Route in SoC Encounter

Now that a gate-level netlist is available, this design can be placed and routed. This is done in Cadence SoC Encounter. The files needed for this process are:

router_top.conf This is the configuration file for Cadence SoC Encounter. It this file, items like liberty, lef and sdc files are specified. They are needed by the place and

route tool.

router_top_unique.v This is the gate-level netlist file out of Design Compiler, used by SoC Encounter for place and route.

NangateOpenCellLibrary_slow.db This is a liberty file in a binary format which can be used by SoC Encounter. In this file timing information is given for all the Nangate standard cells working in slow (worst-case) conditions. The used delay format is *NLDM (Non Linear Delay Model).*

NangateOpenCellLibrary_fast.db This is a liberty file in a binary format which can be used by SoC Encounter. In this file timing information is given for all the Nangate standard cells working in fast (best-case) conditions. The used delay format is *NLDM (Non Linear Delay Model).*

SZAA55_16X37X1CM2_WC.db This is a liberty file in a binary format which can be used by SoC Encounter. In this file timing information is given for the Faraday macro working in WC (worst-case) conditions. The used delay format is *NLDM (Non Linear Delay Model).*

SZAA55_16X37X1CM2_BC.db This is a liberty file in a binary format which can be used by SoC Encounter. In this file timing information is given for the Faraday macro working in BC (best-case) conditions. The used delay format is *NLDM (Non Linear Delay Model).*

router_top_mapped_timing.sdc This is a *standard delay format (SDF)* file with all the design constraints.

NangateOpenCellLibrary.tech.lef This is the LEF file where the technology is described.

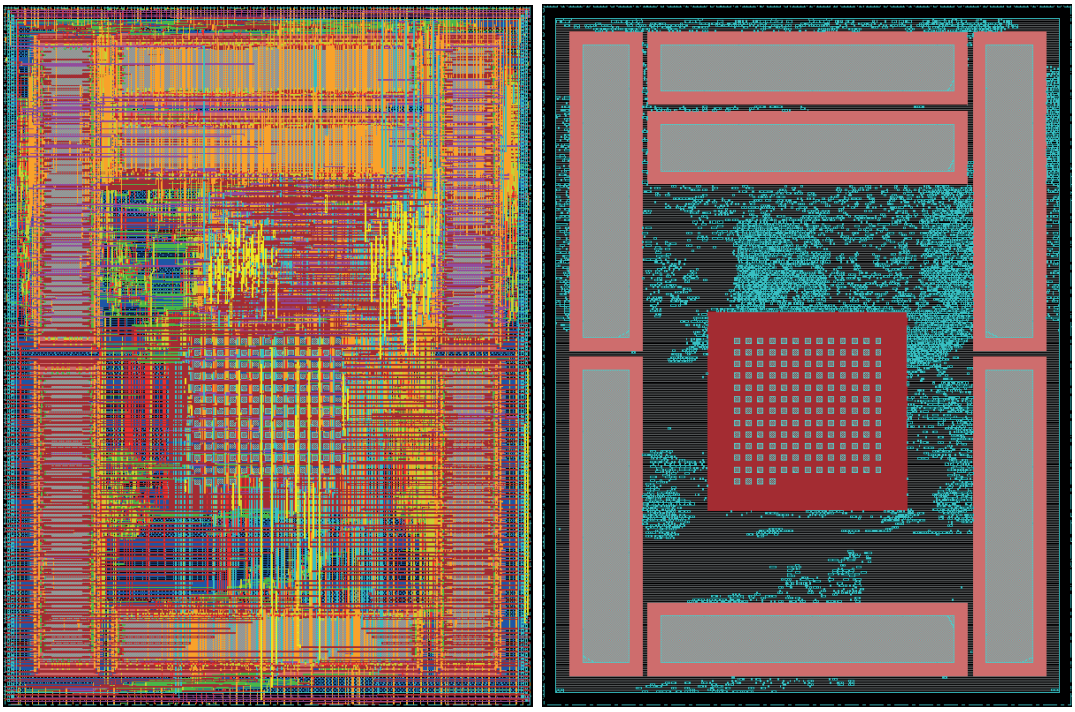
NangateOpenCellLibrary.macro.lef This is the LEF file where all the macros are described.

tsv_in_out.lef This is the LEF file where the TSV is described.

SZAA55_16X37X1CM2.lef This is the LEF file where the memory block is described.

7 TCL files These are the files which specifies all the needed information for the tool, and control it.

Figure 3.9 shows the floorplan of the 3D router in two versions. The floorplan in Figure 3.9a includes the metal wiring, where the floorplan in Figure 3.9b doesn't include the metal wiring. In Figure 3.9b the TSV array can be observed clearly, together with the 7 FIFO blocks. The keep-out-zone of 20 micron around the TSV array can also be seen. The minimum clock cycle time at which the design didn't have a negative slack, was 4 ns, which translates into an operating frequency of 250 MHz. Comparing these nanoseconds of the design with the femtoseconds for the TSV, it can be concluded that the data through the TSVs could be transported much faster. With this in mind, the data through a TSV could be serialized, which will be explained in Chapter 4.



(a) Floorplan with metal wires shown (b) Floorplan without metal wires shown

Figure 3.9: Floorplan of the 3D router

Reducing the data lines in vertical direction by serialization

4

4.1 Introduction

Because the router communicates with 37-bit data flits, various flow control signals, different asynchronous clock signals and a reset signal, a lot of data lines are needed for the communication between two routers. It is inefficient to use a data line for each of the 37 bits for communication. In the current design, one bit is sent through a data line each clock cycle on which the router operates. But a data line has much more capacity, because it's small delay enables the transportation of bits in a much smaller time window. If this effect is exploited, the bits can be sent faster through the data lines. This means also that the same amount of bits can be sent trough a smaller amount of data lines. This is especially interesting for the vertical connections where a TSV is needed per data line. Because the TSV has a large size compared to standard cells, it has a negative impact on the used chip area. If the amount of TSVs can be reduced, it would mean a substantial decrease in occupied area by it. This will be further discussed in Chapter 6. First a way has to be found of how to reduce the number of TSVs, thus reducing the needed vertical data lines.

In Section 3.4.1.3 it is shown that the delay of a signal in a TSV is very small, which results in high speed communication. The communication in a TSV can run at much higher frequencies than the router. This feature can be exploited with the use of serialization. In one slow clock cycle at which the router operates, multiple clock cycles can be used to send smaller amount of data in the same time frame of the slow clock cycle [5]. This is the essence of serialization. Because a smaller amount of data is transported, fewer data lines are needed, which in turn results in fewer TSVs. Because this data needs to be in it's original state when received by the receiver, a deserializer is needed, which puts all the smaller data packets together, where at the end of the slow clock cycle, the original data is retrieved.

Because most of the communication consists of data, they are most suitable to be serialized. The flow control, clock and reset signal are left out, thus are not serialized. The reason for this is that those signals should be valid all the time, else the operation of the router could be compromised. This means that only 37-bits per input or output UP or DOWN port will be serialized.

In the following sections the serializer and the deserializer will be discussed. These two components together enable that the 37-bit communication takes place with the use of fewer data lines, which will result in fewer TSVs and less occupied area. The operation of both components will be explained and how they are implemented in the router design.

For the design of the serializer and deserializer, a design is used from [5]. Here it

is described how the serializer and deserializer are implemented. The only difference is that in the design from [5] it is not used for vertical communication, but for the communication between the CPU and the router. Other aspects are the same.

4.2 Serializer

The input and output signals of the serializer are given in Table 4.1. The basic operation of the serializer is that it divides a chunk of data into several parts of each equal amount of bits, and sends them on each fast clock cycle [5]. The problem in this case is, that our input data is 37-bits wide. That is not a nice number to be divided. A trick has to be used, which will enable an easy division of bits. Three zeros will be padded as the most significant bits in the bit vector, so that a new vector is 40-bits wide. 40 is a good number to be divided, i.e. it can be divided by 2,4,8,10 and 20, which means into parts of 20,10,5,6 and 2 bits. For this thesis, these values are sufficient to see the differences in area and performance with different serialization ratios, which will be shown in Chapter 6.

Table 4.1: Input and output signals serializer

Name	**Description**
clk_fast_i	Fast serialization clock for data
clk_slow_i	Slow router clock at which data is received from the output port
clk_fast_o	Fast serialization output clock
clk_slow_o	Slow router output clock
rst_p_i	Reset signal for the serializer
rst_p_i_pos	Reset signal for the out_phase
parallel_i	37-bit parallel input data
serial_o	$40/R$-bit serial output data

The fast clock frequency is determined by the serialization ratio or R in (4.1).

$$clk\_fast\_i = clk\_slow\_i \times R \tag{4.1}$$

The serializer has internal signals, which are important to understand:

piso_regs: This is an array which consists of $R - 1$ times $SERIAL\_REG\_WIDTH$ bits, where the R is defined in (4.1) and $SERIAL\_REG\_WIDTH$ is defined by $40/R$.

next_piso_regs: This is the same array as the *piso_regs* array, only this one is clocked and is used as registers.

out_phase: This signal is is high when *clk_slow_i* is high, and stays high for a *clk_fast_i* period of time. It is used to flag that the data is ready on the *parallel_i* input.

piso_phase: This signal is the same as *out_phase*, only inverted.

In Figure 4.1 the different steps of serialization are shown, where the serialization ratio R is set on 4. In the rest of the examples this serialization ratio will be used, purely for illustrative reasons. The steps explaining the operation of the serializer are the following:

On T1:
- The clk_fast_i and clk_slow_i clocks goes to high.
- The $piso\_phase$ signal goes to low.
- Data from the 37-bit up-stream output port is put on $parallel\_i(36:0)$.
- Data from $parallel\_i(36:0)$ is put on $parallel\_in\_s(36:0)$. Zeros are put on $parallel\_in\_s(39:37)$.
- Data from $parallel\_in\_s(9:0)$ is put on $next\_piso\_regs[1]$. Data from $parallel\_in\_s(19:10)$ is put on $next\_piso\_regs[2]$. Data from $parallel\_in\_s(29:20)$ is put on $next\_piso\_regs[3]$.
- Data from $parallel\_in\_s(39:30)$ is put on $serial\_o[9:0]$.

On T2:
- The $piso\_phase$ signal goes to high.
- The clk_fast_i clock goes to high.
- The previous data from $next\_piso\_regs$ is clocked to $piso\_regs$.
- Data from $parallel\_in\_s(9:0)$ is put on $next\_piso\_regs[1]$.

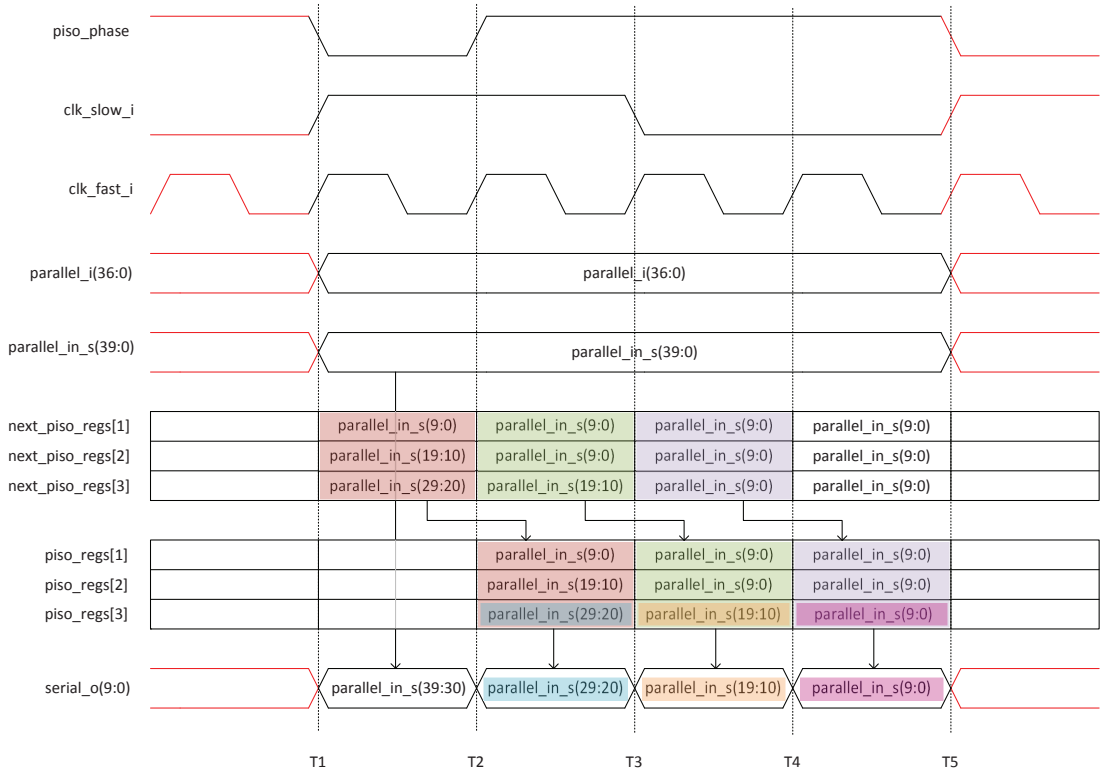


Figure 4.1: Data serialization timing diagram

– Data from $piso\_regs[1]$ is put on $next\_piso\_regs[2]$.
– Data from $piso\_regs[2]$ is put on $next\_piso\_regs[3]$.
– Data from $piso\_regs[3]$ is put on $serial\_o(9:0)$.

On T3 and T4:
– The clk_fast_i clock goes to high.
– The previous data from $next\_piso\_regs$ is clocked to $piso\_regs$.
– Data from $parallel\_in\_s(9:0)$ is put on $next\_piso\_regs[1]$.
– Data from $piso\_regs[1]$ is put on $next\_piso\_regs[2]$.
– Data from $piso\_regs[2]$ is put on $next\_piso\_regs[3]$.
– Data from $piso\_regs[3]$ is put on $serial\_o(9:0)$.

On T5: The same happens as on T1, only with new 37-bit data available on the $parallel\_i$ port.

4.3 Deserializer

The deserializer is the component which is used to change the data back to it's original state, so it can be processed further by the downstream router. The basic operation of the deserializer is that it takes the serialized bits of data putting them back on each fast clock cycle in a new data vector, so the original data is retrieved [5]. Because the data which is serialized is 40 bits wide, the three zeros which were initially padded as most significant bits, they have to be subtracted so the original data is retrieved. The input and output signals of the deserializer are given in Table 4.2.

Table 4.2: Input and output signals deserializer

Name	**Description**
clk_fast_i	Fast deserialization clock for data
clk_slow_i	Slow router clock at which data is received from the upstream output port
clk_slow_o	Slow router output clock
rst_p_i	Reset signal for the deserializer
serial_i	$40/R$-bit serial input data
parallel_o	37-bit parallel output data

The deserializer has internal signals, which are important to understand:

sipo_regs: This is an array which consists of $R-1$ times $SERIAL\_REG\_WIDTH$ bits, where the R is defined in (4.1) and $SERIAL\_REG\_WIDTH$ is defined by $40/R$. This array is clocked and used as registers.

In Figure 4.2 the different steps of deserialization are shown, where as in Section 4.2 the serialization ratio R is set on 4. In the rest of the examples this serialization ratio will be used, purely for tentative reasons. The steps explaining the operation of the deserializer are the following:

On T1:
– The clk_fast_i and clk_slow_i clocks goes to high.

– Data from $serial\_i(9:0)$ is put on $parallel\_out\_s(9:0)$.
– Data from $parallel\_out\_s(9:0)$ is put on $parallel\_o(9:0)$.

On T2:
– The clk_fast_i clock goes to high.
– Data from $serial\_i(9:0)$ is put on $parallel\_out\_s[9:0]$.
– The previous data from $serial\_i(9:0)$ is clocked to $sipo\_regs[1]$.
– Data from $sipo\_regs[1]$ is put on $parallel\_out\_s(19:10)$.
– Data from $parallel\_out\_s(19:0)$ is put on $parallel\_o[19:0]$.

On T3:
– The clk_fast_i clock goes to high.
– Data from $serial\_i(9:0)$ is put on $parallel\_out\_s(9:0)$.
– The previous data from $serial\_i(9:0)$ is clocked to $sipo\_regs[1]$.
– The previous data from $sipo\_regs[1]$ is clocked to $sipo\_regs[2]$.
– Data from $sipo\_regs[1]$ is put on $parallel\_out\_s(19:10)$.

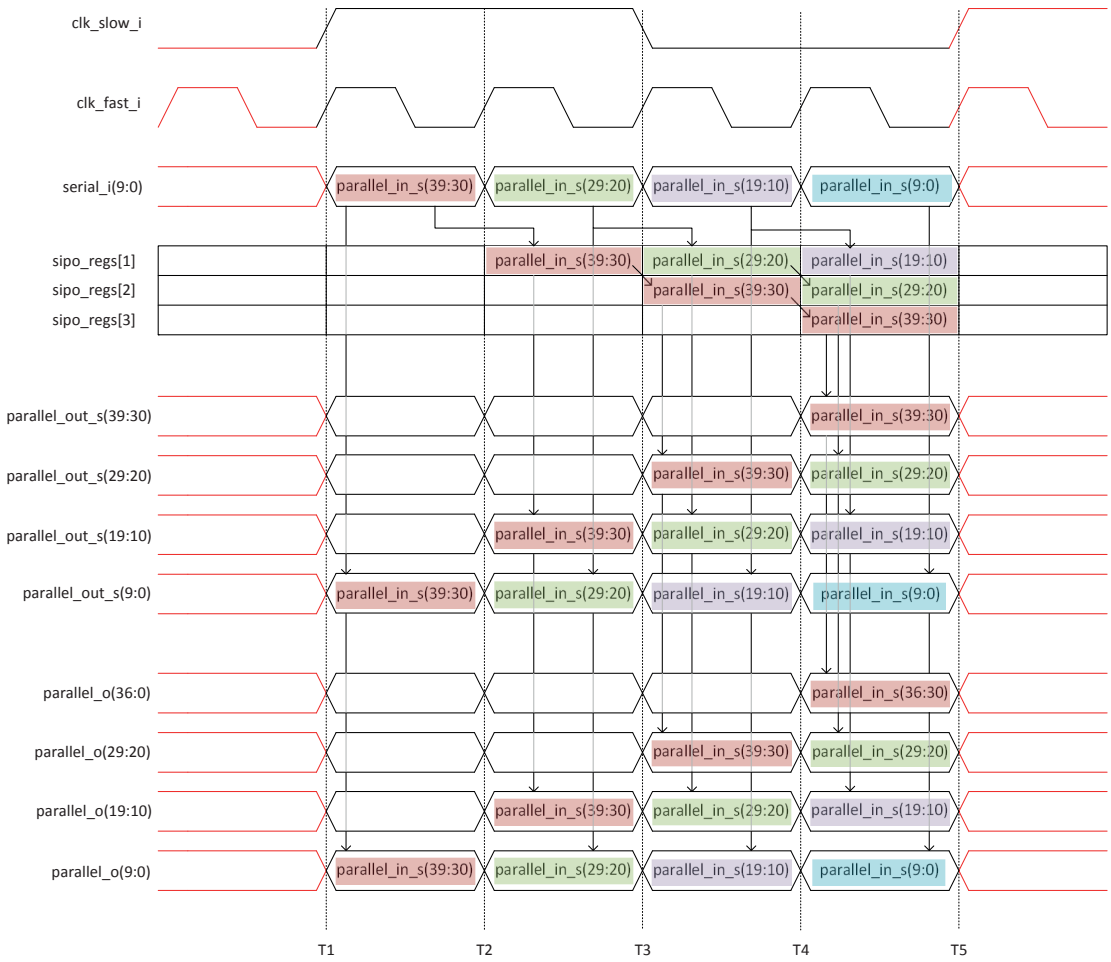


Figure 4.2: Data deserialization timing diagram

– Data from $sipo\_regs[2]$ is put on $parallel\_out\_s(29:10)$.
– Data from $parallel\_out\_s(29:0)$ is put on $parallel\_o(29:0)$.

On T4:
– The clk_fast_i clock goes to high.
– Data from $serial\_i(9:0)$ is put on $parallel\_out\_s(9:0)$.
– The previous data from $serial\_i(9:0)$ is clocked to $sipo\_regs[1]$.
– The previous data from $sipo\_regs[1]$ is clocked to $sipo\_regs[2]$.
– The previous data from $sipo\_regs[2]$ is clocked to $sipo\_regs[3]$.
– Data from $sipo\_regs[1]$ is put on $parallel\_out\_s(19:10)$.
– Data from $sipo\_regs[2]$ is put on $parallel\_out\_s(29:20)$.
– Data from $sipo\_regs[3]$ is put on $parallel\_out\_s(39:10)$.
– Data from $parallel\_out\_s(36:0)$ is put on $parallel\_o(36:0)$.

On T5: The same happens as on T1, only with new 10-bit data available on the $serial\_i$ port.

4.4 Implementation and timing

When looking at Figure 4.1 and Figure 4.2 all the clocks are nicely synchronous without any clock or data delay. This example is actually based on zero-delay simulation. The largest delay is between the serializer output and deserializer input, because they are connected with TSVs. In Section 3.4.1.3 it is shown that the delay in TSVs is not that big.

When implementing this into a real design like ASIC or *Field Programmable Gate Array (FPGA)*, these delays will impose timing problems. In case of an ASIC, these two routers could be put on separate boards, where the timing is substantial. Even in a 3D ASIC design, what this project is about, timing could impose some communication problems. Some timing constraints are important to obey, in order to let this design work. Even if the delay between the serializer and deserializer is large.

In figure Figure 4.3 the timing diagram of the serializer and deserializer is shown with delayed signals, where T_s is the slow clock cycle, T_f is the fast clock cycle, t_{cds} is the delay of the slow clock, t_{cdf} is the delay of the fast clock, t_{dmin} is the minimum data delay and T_{dmax} is the maximum data delay.

The following equations describe the timing between the serializer and deserializer, where R is the ratio which is used for serialization and α is a factor where the clock behavior is repeating it self.

$$0 \leq t_{cdf} < t_{dmin} \tag{4.2}$$

$$t_{dmax} + (\alpha - 1)T_f < t_{cdf} < t_{dmin} + \alpha T_f \qquad \text{for } \alpha = 1, 2, 3, \ldots \tag{4.3}$$

$$0 \leq t_{cds} \leq t_{cdf} \tag{4.4}$$

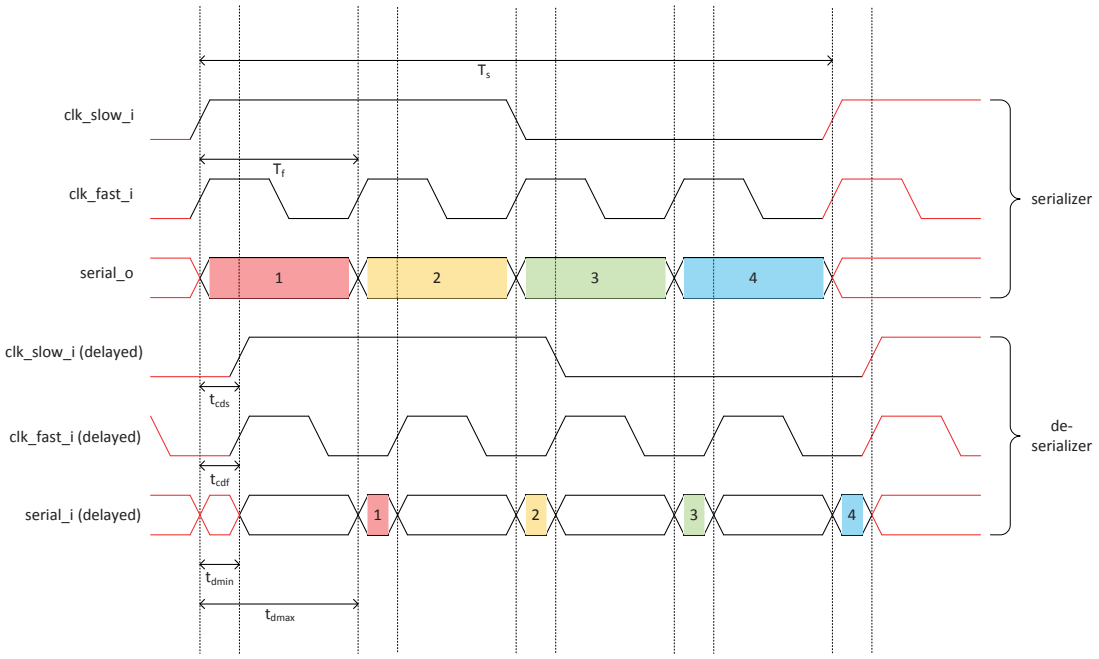


Figure 4.3: Timing diagram serializer/deserializer delayed siganls

$$t_{dmax} + (\alpha R - 1)T_f < t_{cds} \leq t_{cdf} + (\alpha R - 1)T_f \qquad \text{for } \alpha = 1, 2, 3, \ldots \tag{4.5}$$

The fast clock delay t_{cdf} should obey (4.2) or (4.3). If t_{cdf} obeys (4.2), the slow clock delay t_{cds} should obey (4.4) or (4.5). If t_{cdf} obeys (4.3), the slow clock delay t_{cds} should only obey (4.5).

(4.2) describes that if the fast clock delay is higher than 0 and lower than the minimum data delay, the data which it clocks is valid. If the fast clock delay is higher than the minimum data delay, it will clock false data in, because not all the bits are valid after the minimum data delay. (4.3) describes that if the fast clock delay is higher than the maximum data delay and lower than the minimum data delay + fast clock cycle, the data which it clocks is valid. If the fast clock delay is lower then the maximum data delay, it will clock false data in, because not all the bits are valid before the maximum data delay. If the fast clock delay is higher than the minimum data delay + fast clock cycle, some bits of the next data have already arrived, and false data is clocked in. (4.4) describes that if the slow clock delay is higher then 0 and lower then the fast clock delay, the data which it clocks is valid. If the slow clock delay is higher than the fast clock delay, it will clock false data in, because a register too many is clocked in and shifted. (4.5) describes that if the slow clock delay is higher than the maximum data delay + $(R-1)$ fast clock cycles and lower than fast clock delay + $(R-1)$ fast clock cycles, the data which it clocks is valid. If the slow clock delay is lower then the maximum data delay + $(R-1)$ fast clock cycles, it will clock false data in, because not all the bits are valid before the maximum data delay. If the slow clock delay is higher than the fast clock delay + $(R-1)$ fast clock cycles, it will clock false

data in, because a register too many is clocked in and shifted.

For equations with factor α in it, all positive integers can be used, because it specifies with how many fast clock cycles the fast clock is delayed, and exactly copied. The use of the R factor is needed to continue the specific relation between the fast clock and the slow clock.

The asynchronous working of the router

5

5.1 Introduction

The communication in this design should be handled asynchronously, which means the routers should operate asynchronously with respect to each other. The main reason for this is that the clock frequency of the different routers can't be exactly the same. This is mostly noticeable between two routers sitting on two different tiers. Another reason is that the communication through the 3D-NoC should be flexible. The reasoning for this is that different injectors (computational units) should be able to work on different frequencies, and still be able to communicate with each other through a 3D-NoC.

A way has to be found how to implement the asynchronous behavior of one router, and let a complete mesh communicate with each other at different frequencies. In this chapter it will be explained how the router of S. Kumar [6] is changed in order to support the asynchronous working. In this process, various problems will be tackled which were encountered during the design of the asynchronous router.

5.2 Design of an asynchronous router

5.2.1 Clocks and resets

First the different clocks have to be determined. For all the clocks it should be known where they are coming from, where they are going to, which part of the design they are clocking, and what the dependency between them is.

Second, the reset signals have to be determined. For all the resets it has to be known where they are coming from, which part of the design they are resetting, and on which clock they are clocked on.

In Appendix B a diagram is given for the complete 3D router. The components and signals are only shown for the LOCAL and UP input/output port. This is enough to be able to determine the operation of the router. This will be needed for the following sections where the clock and reset signals will be described.

5.2.1.1 Clocks

A 3D router has 7 input ports, at which the data is coming in at a particular clock frequency, and is written into the memory. This means that every input port has it's own frequency. In two out of the seven ports, the UP and the DOWN port, the case is a little bit different. Here the clocks are not used directly for the input port, but for the deserializer, as seen in Chapter 4. All the input clocks are given in Table 5.1.

There are also 7 output arbiters available, at which the data is read from the memory and at which the data comes out at a particular frequency. This frequency is the same

Table 5.1: Input clocks asynchronous router

Name	Description
CLK_f_U_i	Fast clock for de-serializer UP port
CLK_s_U_i	Slow clock for de-serializer UP port
CLK_f_D_i	Fast clock for de-serializer DOWN port
CLK_s_D_i	Slow clock for de-serializer DOWN port
CLK_s_N_i	Slow clock for writing input NORTH port
CLK_s_S_i	Slow clock for writing input SOUTH port
CLK_s_E_i	Slow clock for writing input EAST port
CLK_s_W_i	Slow clock for writing input WEST port
CLK_s_L_i	Slow clock for writing input LOCAL port

for all the output arbiters. It's called the router frequency. This is the frequency at which the router sends the data to a different router or other component. Even if all the output arbiter clock frequencies are the same for one router, all the output arbiters send this clock off individually, which can be used for the downstream input port. This is done because of clocking reliability. Only output clock frequencies from the UP and DOWN serializer are not sent directly to the output, as seen in Chapter 4, but are first used for the serializer. The slow clock for a particular serializer comes from the output arbiter, where the fast clock comes from the input of the router. Both these clocks are propagated to the output of the router, which can be used for the input of a downstream router. All the output clocks are given in Table 5.2.

Table 5.2: Output clocks asynchronous router

Name	Description
CLK_f_U_o	Fast clock from the serializer UP port
CLK_s_U_o	Slow clock from the serializer UP port
CLK_f_D_o	Fast clock from the serializer DOWN port
CLK_s_D_o	Slow clock from the serializer DOWN port
CLK_s_N_o	Slow clock from the output arbiter NORTH port
CLK_s_S_o	Slow clock from the output arbiter SOUTH port
CLK_s_E_o	Slow clock from the output arbiter EAST port
CLK_s_W_o	Slow clock from the output arbiter WEST port
CLK_s_L_o	Slow clock from the output arbiter LOCAL port

Next to the mentioned clocks, there is a router slow clock, which is used for all the input ports when reading out the data from the memory. The same clock is used for all the output arbiters. Together with this router slow clock, there is a router fast clock, which is a factor R faster then the router slow clock and is synchronous to the router slow clock. The router fast clock is needed for the serialization of the data coming out of the output arbiter, as seen in Chapter 4. These clocks are given in Table 5.3.

Because the communication between two routers is dependent on the clock, it is important that the same clock is used for reading out the router, and writing to another

Table 5.3: Global clocks asynchronous router

Name	Description
CLK_i	Fast clock for the router
CLK_i_slow	Slow clock for the router

downstream router. To accomplish this, some intermediate clocks are needed. These clocks are given in Table 5.4.

Table 5.4: Intermediate clocks asynchronous router

Name	Related to	Description
s_clk_f_U_i	CLK_f_U_i	Fast clock de-serializer UP from input TSV
s_clk_s_U_i	CLK_s_U_i and CLK_W_U_i	Slow clock de-serializer UP from input TSV
s_clk_f_D_i	CLK_f_D_i	Fast clock de-serializer DOWN from input TSV
s_clk_s_D_i	CLK_s_D_i and CLK_W_D_i	Slow clock de-serializer DOWN from input TSV
CLK_W_U_i	s_clk_s_U_i	Slow clock writing input UP from deserializer
CLK_W_D_i	s_clk_s_D_i	Slow clock writing input DOWN from deserializer
CLK_U_o	CLK_i_slow and s_clk_s_U_o	Slow clock serializer UP from output arbiter
CLK_D_o	CLK_i_slow and s_clk_s_D_o	Slow clock serializer DOWN from output arbiter
s_clk_f_U_o	CLK_i and CLK_f_U_o	Fast clock output TSV from serializer UP
s_clk_s_U_o	CLK_i and CLK_s_U_o	Slow clock output TSV from serializer UP
s_clk_f_D_o	CLK_i and CLK_f_D_o	Fast clock output TSV from serializer DOWN
s_clk_s_D_o	CLK_i and CLK_s_D_o	Slow clock output TSV from serializer DOWN

All these clocks should be categorized into synchronous clock groups. All clocks in one group are synchronous to each other, but the clocks from one group are asynchronous to clocks from another group. This way, the synthesis tool knows how to compute the timing paths. The timing paths are computed between the same clock domains. Whenever there is a *Clock Domain Crossing (CDC)*, the timing path is set to false. So it is excluded from the timing analysis. In Table 5.5 all the clock groups are shown.

Table 5.5: Clock groups

Group	Clocks
I	CLK_i,CLK_i_slow
II	CLK_f_U_i,CLK_s_U_i
III	CLK_f_D_i,CLK_s_D_i
IV	CLK_s_N_i,
V	CLK_s_S_i,
VI	CLK_s_E_i,
VII	CLK_s_W_i,
VIII	CLK_s_L_i,

5.2.1.2 Resets

Every design needs a reset signal. A reset ensures that the system can go into it's initial state at startup. There are a couple of reset signals in this design. Some of these are global and some are local. All the resets are given in Table 5.6.

Table 5.6: Reset signals

Name	Component	Clocked on	Used for
RST_p_i	ROUTER_TOP	-	All reset signals
RST_p_i	INPUT_PORT	Negative edge read clock	Read processes
RST_p_w_i	INPUT_PORT	Negative edge write clock	(Initialization) write reset
RST_p_r_i	INPUT_PORT	Negative edge read clock	Initialization read reset
S_RST_WE	INPUT_PORT	Positive edge write clock	Initialization write address memory
S_RST_RE	INPUT_PORT	Positive edge read clock	Initialization read address memory
RST_p_i	OUTPUT_ARBITER	Negative edge read clock	Read processes
RST_p_i	SERIALIZER	Negative edge read clock	Filling registers with zeros
RST_p_i_pos	SERIALIZER	Positive edge read clock	Setting counter initial state
RST_p_i	DESERIALIZER	Negative edge read clock	Filling registers with zeros

Because the router has to be able to reset itself, a reset is needed which will facilitate this. The main component of the router is it's FIFO, which is made of dual-port memory. The memory doesn't have it's own reset signal, which means a way has to be found how to bring the memory into it's initialization state. In the design of S. Kumar [6], a mechanism is used where when the reset signal gets high, with the first rising clock edge, an initialization reset signal is triggered. This reset signal is used for the writing reset. On the following rising clock edge, this reset is propagated to the read reset. This ensures that first the write reset is getting high, after which the read reset is getting high. The write reset signal lets the memory write zeros into it's first address, where after one clock cycle those zeros are read from the first address of the memory. This process is the actual initialization of the memory.

The problem with this design is that it's using one clock for reading and writing. When using two asynchronous clocks, some adjustments have to be made. First of all two different reset signals have to be made, which should be clocked on the write and the read clock. Another condition is that the read reset can go into high state, after the write reset went into high state, and zeros are written into the first address of the memory. Only when this happens, the read reset can go into high state, which will ensure that the zeros from the first address are being read out. The initialization write and read reset signals are shown in Figure 5.1. Here CLK_W_i and CLK_R_i are the write and the read clock of the FIFO. RST is the global reset, $RST\_p\_w\_i$ and $RST\_p\_r\_i$ are the resets used for the memory initialization. S_RST_WE and S_RST_RE are the initialization resets. The steps describing the generation of S_RST_WE and S_RST_RE are:

1. At T1 the global reset RST goes high.
2. At T2 on the falling edge of CLK_W_i, reset $RST\_p\_w\_i$ goes to high.

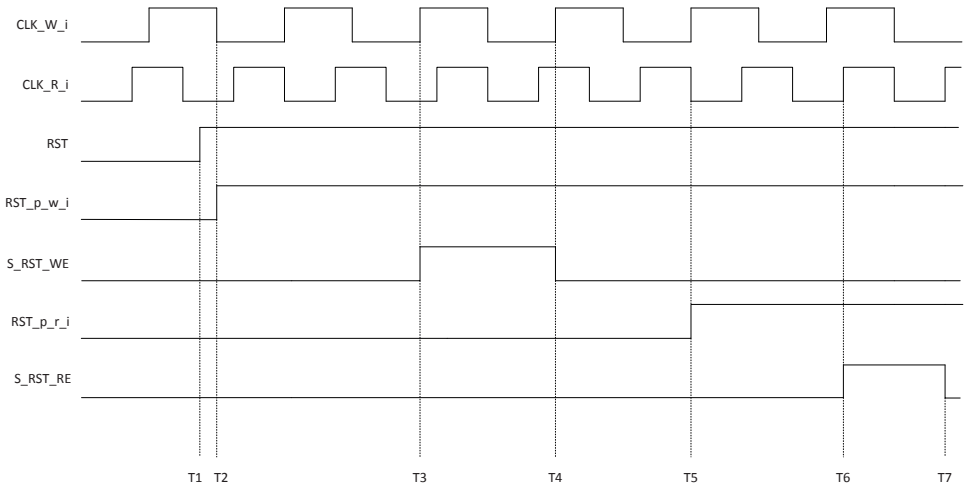


Figure 5.1: The making of the initialization write and read resets

3. After two rising clock edges of CLK_W_i at T3, reset S_RST_WE goes to high, and stays high for one write clock cycle until T4. In this time zeros are written in the first address of the memory.

4. After $RST\_p\_w\_i$, two write cycles are needed to ensure that the S_RST_WE and that the writing of zeros into the first memory address if finished. Additional two read cycles are needed to synchronize the reset to the read clock CLK_R_i, and at T5 reset $RST\_p\_r\_i$ goes to high.

5. After two rising clock edges of CLK_R_i at T6, reset S_RST_RE go to high, and stay high for one write clock cycle until T7. In this time zeros are written in the first address of the memory.

The write reset $RST\_p\_w\_i$ is further used for the synchronization of the read address $S\_R\_ADR\_I$ into the write clock, which will be discussed in Section 5.2.2. It is also used to determine if the input at the input port is valid, and for the write process of the data into the memory.

The read reset RST_p_i is used for the synchronization of the write address $S\_R\_ADR\_I$ into the read clock, which will be discussed in Section 5.2.2. It is also used to set the state machine of the input port to it's initial state, and for the read process of the data from the memory.

5.2.2 Synchronization

5.2.2.1 FIFO

In the input port CDC occurs when the data comes in (written to) at one frequency, and comes out (read from) at another frequency. A solution for this is found by S. Kumar [6], by using a FIFO. A FIFO uses dual-port memory, which writes data at one frequency and reads data at another frequency, without any synchronization problems.

But a FIFO doesn't consist only from the dual-port memory. It has added functionality in the form of FIFO-full and FIFO-empty signals. These signals are used to let the sender know, when the memory is full, and the memory should stop writing. Otherwise it is used to let the memory stop reading in case the memory is empty. This signals are important in order to ensure that the complete functionality of a FIFO is working.

The first thing to do in designing a good FIFO, is to determine to which clocks the signals are clocked, and how these signals interact with each other. These signals are the following:

S_WE_FAR This signal enables the writing to a memory address when S_WE_I or S_RST_WE is high. Because those two signals are both clocked on the write clock, S_WE_FAR is also clocked on the write clock and no CDC occurs.

S_RE_FAR This signal enables the reading from a memory address when S_RE_I and not S_RST_WE is high or when S_RST_RE is high. Because those two signals are both clocked on the read clock, S_RE_FAR is also clocked on the read clock and no CDC occurs.

S_WE_I This signal is high when the input data is not zero, and the write reset is not high. This signal is clocked on the write clock.

S_RE_I This signal is high when the router logic has made a connection to a valid output port. This signal is dependent on S_DAT_READY, so also dependent on it's clock. In this case this is the read clock.

S_RST_WE This reset signal initializes the memory by writing zeros into the first address. It does that on the write clock after the write reset is high.

S_RST_RE This reset signal initializes the memory by reading zeros from the first address. It does that on the read clock after the read reset is high. The read reset can get high when the write reset is high, and has written the zeros into the first memory address. In this way it is ensured that the memory will read from it's address after this address has been filled with zeros. This way the memory is initialized.

S_DAT_READY This signal determines if the FIFO is empty. This signal depends on $S\_NEXT\_R\_ADR$ and $S\_W\_ADR\_I$, which are both clocked at different frequencies. Some synchronization has to be done to one of those signals.

S_FIFO_FULL This signal determines if the FIFO is full. This signal depends on $S\_W\_ADR\_I$ and $S\_R\_ADR\_I$, which are both clocked at different frequencies. Some synchronization has to be done to one of those signals.

S_NEXT_R_ADR This is the next address which will be read from the memory. This signal is clocked on the read clock.

S_W_ADR_I This is the write address at which the data is written into the memory. This signal is clocked on the write clock.

S_R_ADR_I This is the read address at which the data is read from the memory. This signal is clocked on the read clock.

The first signal where CDC is occurring, is the S_DAT_READY. This signal determines if the FIFO is empty. It does that by comparing the next read address $S\_NEXT\_R\_ADR$ to the write address $S\_W\_ADR\_I$. When the read address catches up with the write address, it means that the FIFO is empty. The problem in this comparison is that both the addresses are clocked on two different clocks, so CDC occurs. Because this check should be performed in the read clock domain, somehow the write address should be synchronized from it's write clock, to the read clock domain.

The second signal where CDC is occurring, is the S_FIFO_FULL. This signal determines whether the FIFO is full. It does that by comparing the write address $S\_W\_ADR\_I$ to the read address $S\_R\_ADR\_I$ with a threshold factor, determined for this particular router design. When the difference between the write address and the read address reaches a certain threshold value, it means that the FIFO is full. Also here problems occur because both these addresses are clocked on different clocks and CDC occurs. Because this check should be performed in the write clock domain, somehow the read address should be synchronized from it's read clock, to the write clock domain.

5.2.2.2 Synchronizer

Because there are signals which have to cross different clock domains, synchronizers are needed. First it has to be explained what can go wrong without a synchronizer, and how a synchronizer tries to solve this problem. We look at a signal that is trying to cross from one clock domain into another. This happens between two flip-flops. Usually the time available for getting the signal from the clocking of one flip-flop, to the clocking of the second flip-flop, is exactly one clock cycle. It is important to remember that every flip-flop has it's own setup and hold time. The setup time is the time at which the signal should arrive before the clock cycle of that particular flip-flop, and shouldn't change values. The hold time is the time after the clock of that particular flip-flop, where the signal cannot change it's value. This is necessary because there is a possibility the signal is not stable enough to be clocked into the flip-flop, and will go into a metastable state. When in this state, it takes some time for the signal to settle down to one value. Every flip-flop has it's own settling time, and it's used to calculate the *Mean Time Between Failures (MTBF)*. The MTBF is given in (5.1) [27], where τ is the settling time constant of the flip-flop, T_W a parameter related to its time window of susceptibility, f_A the synchronizer's clock frequency, and f_D is the frequency of the incoming data. The MTBF is designed to be much longer (in the order of 10x) than the expected lifetime of the circuit [27]. It can be seen in (5.1) that the smaller the settling time constant, the longer the MTBF. However, this dependent on the flip-flop and it's technology. If the signal goes into metastable state and is propagated further into the circuit, it will cause a malfunction in the circuit.

$$MTBF = \frac{e^{\frac{T}{\tau}}}{T_W f_A f_D} \tag{5.1}$$

To solve this metastable state of the signal, it is necessary to give the metastable output signal enough time to settle down to a stable value in the destination clock domain [28]. At the basis of most synchronizers is a two-flop synchronizer, which is shown in Figure 5.2. The picture shows 3 flip-flops, where the first one is clocked on

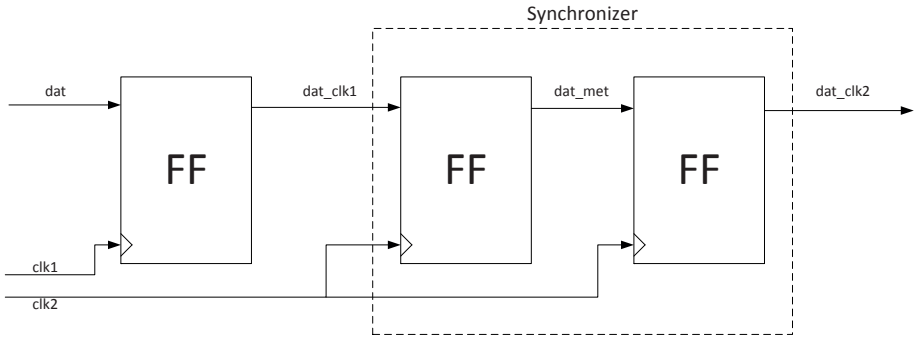


Figure 5.2: Two flip-flop circuit

the $clk1$ source clock, and the second two are clocked at the $clk2$ destination clock. The actual synchronizer is composed of the second two flip-flops. When the asynchronous dat_clk1 signal is captured by the first synchronizer flip-flop into the destination clock domain, it waits for a full destination clock cycle to let any metastability in the dat_met signal to settle to a valid value. After that clock cycle, the dat_met signal is hopefully settled and at the next destination clock cycle it is sampled into the second flip-flop [28]. This way the dat_clk2 signal is the synchronized version of the dat_clk1 signal. Note that in theory, the one cycle given the metastable signal to decay to one value, may not by enough, and this synchronization technique won't work. But with the help of the MTBF a designer can calculate the probability of this going wrong, and if this probability is small enough, it may be sufficient for the design to work in it's projected lifetime.

Back to the FIFO, there are some signals, which have to be synchronized, in order to let the FIFO, and the router work properly. For this, the two-flop synchronizer can be used. However, there is another problem. The address signals which have to be synchronized are all multiple-bit signals. It is plausible to think that if for one signal one two-flop synchronizer is needed, for multiple signals multiple two-flop synchronizers are needed. This is actually true, but not sufficient. In the case of the address signals, which are binary, and which are incremented by one, it is possible for this signal to change multiple bits at once. When trying to synchronize these simultaneous changing bits, there is a possibility that one bit will go into metastable state, another won't, one will come out of it's metastable state, and another won't. This unpredictability is disastrous for the circuit as it will propagate invalid signals. An example is given in Figure 5.3. There are three address bits, from which two are changing on the positive clock edge. Signals $ADR[2]$ and $ADR[0]$ both are getting high on the positive clock edge. Signal $ADR\_sync[2]$ goes into metastable state and eventually settles down at a high value. Signal $ADR\_sync[0]$ also goes into metastable state, but this signal settles

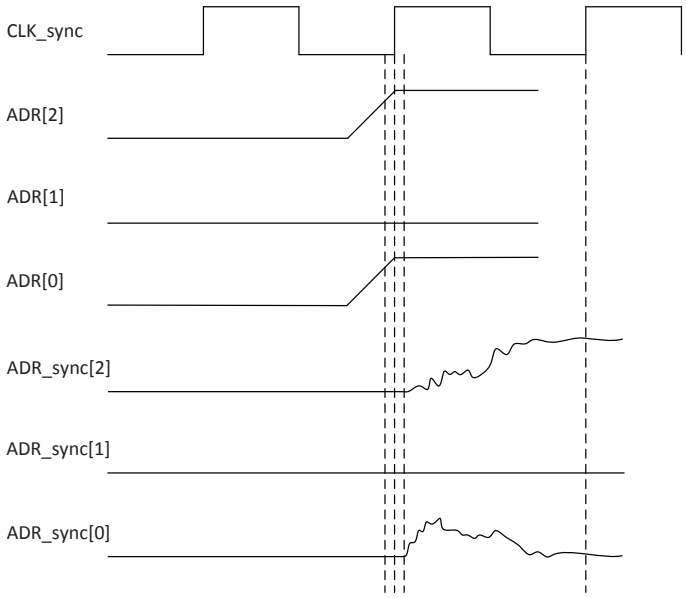


Figure 5.3: Example of data incoherence

down at a low value. $ADR[1]$ doesn't change thus cannot become metastable. Looking at the address bit vector, '101' is the input and '100' is the synchronized output, which is invalid.

To eliminate the problem of trying to synchronize multiple changing signals on the same clock edge, Gray Coding can be used [7, 28]. The difference between Gray Code and Binary code is, that a Gray Code signal can change only one bit at a time, as the Binary signal can change multiple bits at the time, the latter of which will impose a problem as described above. When changing only one bit at a time, it is only possible that one bit will have to be synchronized into the other clock domain. This way only two possibilities exist. The synchronized signal will retain it's old value, of it will propagate it's new value. There is no chance that an invalid state of the signal will be propagated. In order to encode a binary signal into a Gray encoded signal, a Gray Encoder is needed. A Gray decoder is needed for the decoding of the Gray encoded signal back to a binary value.

The Gray encoder/decoder together with the synchronizer is incorporated into the FIFO and is shown in Figure 5.4. To compare the two address signals with each other, first the signal which has to be synchronized, should be Gray encoded. After this, the signal is synchronized, and at the end, the synchronized signal is decoded back into binary for comparison with the other address signal. This way the FIFO full and FIFO empty states are safely determined.

Using synchronization from a fast to slow clock domain imposes a problem where not all of the binary values can be passed to a new clock domain [7]. The question is, if it is important whether the binary value keeps incrementing and overflowing or underflowing the FIFO between the sampled values. When looking at the FIFO full, this occurs when the write pointer catches up to the synchronized and sampled read

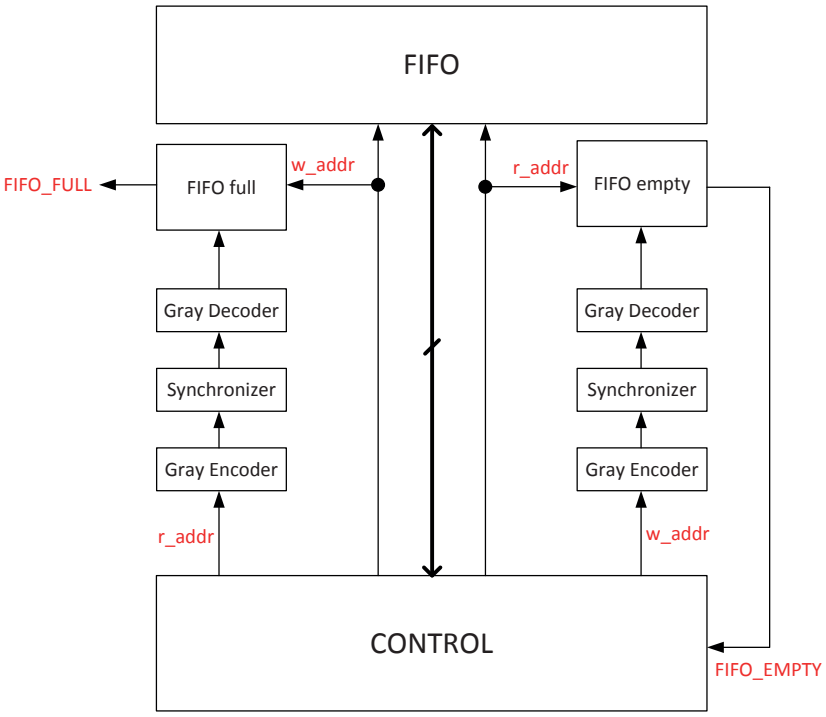


Figure 5.4: Gray encoder/decoder & synchronizer incorporated into FIFO

pointer. The synchronized and sampled read pointer might not reflect the current value of the actual read pointer, but the write pointer will not try to count beyond the synchronized read pointer value, thus overflow will not occur [7]. The same theory applies on the FIFO empty, because the read pointer will not try to count beyond the synchronized write pointer value [7]. This way of determining the FIFO full and FIFO empty states could be seen as pessimistic, but in the end it's a safe method to use.

5.3 A note about simulation of synchronizers in ModelSim

In Section 5.2.2.2 it is explained that the basic component of a synchronizer consists of two flip-flops connected to each other as shown in Figure 5.2. In real life the input signal which changes close to the clock edge of the first flip-flop can go into metastable state and settle down before the next clock cycle, which then in turn will be clocked in by the second flip flop. When simulating synthesized and/or placed&routed code with real components and timing in a HDL simulator like ModelSim, the example above will result in setup/hold violations, and an unknown signal[1] will be propagated into the second flop. It will not come out of that unknown state as in real life, but will just be clocked in by the second flip-flop after the second clock edge. In the end, this unknown

[1] An unknown signal will be shown as a red signal in ModelSim. This signal can not be defined by the simulator.

signal will be used further on in the logic, and the circuit behavior will be broken.

The solution for this simulation problem is to get rid of the timing of the first flip-flop of every synchronizer. A way to do this is to adjust the sdf file, where the timing of all the instances is defined. The setup and hold timing should be changed into zero. This way, the first flip-flop will always propagate the right input value to the second flip-flop.

6 Application and Results

This chapter describes the results of two implementations. The first one is an RTL implementation where two routers are connected to each other, and the second is a placed and routed implementation of a single router. In Section 6.1 results are given of the throughput and latency of the two routers, where in Section 6.2 timing and area results are given for the single router.

6.1 2 routers in RTL

6.1.1 Design

To test the serialization, together with the asynchronous behavior, 2 routers should be connected through their vertical interconnections, where data is injected at 1 frequency, router 1 operates at another, and router 2 operates at a third. The serializer/de-serializer is designed in such a way, that is has no influence on the throughput and latency of the two connected routers. This is explained in Chapter 4. What can be tested is the logical behavior of the serializer/de-serializer.

In Figure 6.1 the 2 routers are shown, together with their interconnections and their associated signals. In every router block there is a de-serializer included at a vertical input port and a serializer included at a vertical output port. Also the logic for asynchronous working of the router is included.

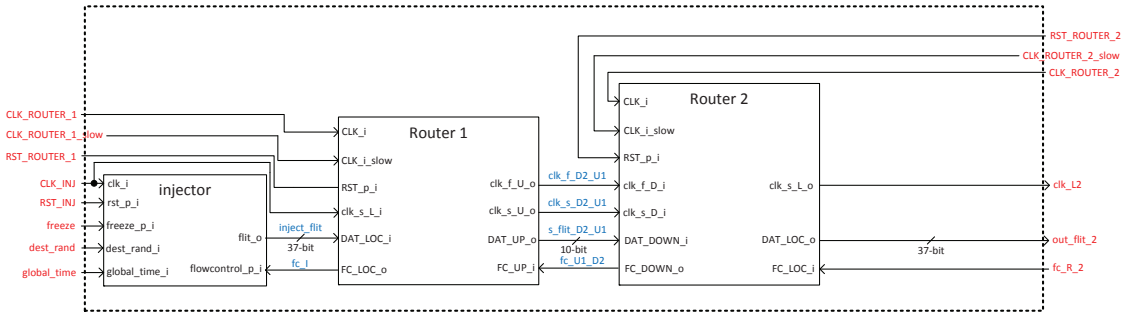


Figure 6.1: 2 routers vertically connected

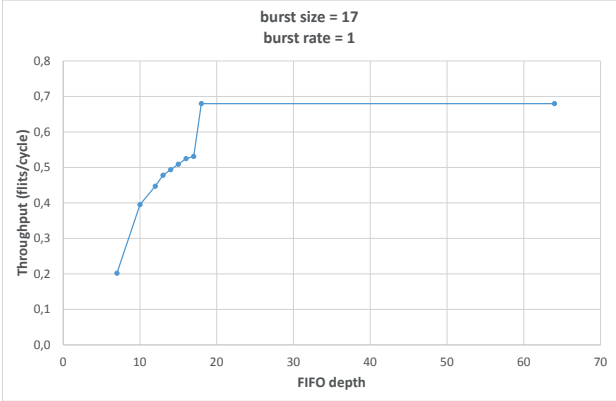
6.1.2 Throughput

The throughput is determined for various interesting cases, where the FIFO depth is changed, the burst size and the burst rate are changed, and the frequencies are changed

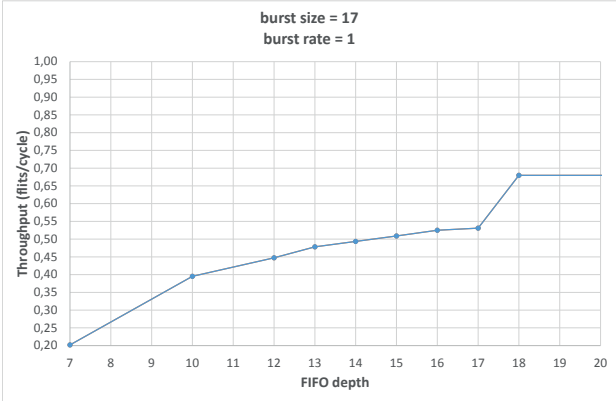
to test an asynchronous environment. On every injector clock cycle, 1 37-bit flit can be injected. If a packet is 17 flits long, it means that it takes a minimum of 17 injector clock cycles to inject 1 packet into a router. A burst size means how many flits are injected in sequence. The best way to test this behavior, is to change the packet size. In this experiment the burst size is equal to the packet size. If the burst size is 3, it means that 1 packet of 3 flits is injected. If the burst size is 8, it means that 1 packet of 8 flits is injected. A burst rate means at which rate the packet, with a specified packet size, can be injected. If the burst rate is 1, it means that the packets with the specified packet size can be injected one after another. If the burst rate is 1/8, it means that after one packet, there is a delay which consists of the amount of time 7 packets to be injected. After that, a new packet is injected.

First an ordinary case is tested where the injector, and both routers operate at the same frequency. On every clock cycle the injector is injecting 1 flit into the input LOCAL port of the first router, which routes the flits into it's output UP port. This port is connected to the input DOWN port of the second router, where the data is propagated to it's output LOCAL port. This test is done with different values of the FIFO depth, to see how this influences throughput. The FIFO depths tested are between 7 and 64. The reason why 7 is the smallest FIFO depth tested, is that the router architecture doesn't support smaller values. The packet size is kept at 17 flits. The result is shown in Figure 6.2. Figure 6.2a shows that the throughput reaches it's maximum at the FIFO depth of 18 and stays at that value until the maximum FIFO depth of 64 is tested. Figure 6.2b is the zoomed in version which excludes the FIFO depth of above 18, because it doesn't change. The maximum throughput obtained lies around 0.67 flits/cycle. This maximum throughput is constrained by the router architecture of S. Kumar. This has to do with the fact that there is a router overhead time for each packet which has to be routed. This ensures that the throughput can never be 1 flit/cycle. Increasing the FIFO depth beyond a certain point, which in this case is 18, doesn't help because the router overhead time still exists. The reason why the throughput jumps up between the FIFO depth of 17 and 18, is that from that FIFO depth one, one whole packet can fit into the FIFO. This means the time a packet has to wait to be injected is minimized.

Figure 6.3 shows the throughput when the burst rate is 1/2, with dependence on burst size or FIFO depth. In Figure 6.3a the case is shown where the FIFO depth is kept at 12 (same as S. Kumars findings), the burst rate is set on 1/2 and the burst size is changed to see what influence this has on the throughput. It is tested for a burst size between 3 and 64. It can be observed that the throughput oscillates up until a burst size of 23, after which it keeps rising slowly. The different peaks in throughput have to do with the fact that packets with particular sizes can fit more easily into the FIFO than other packets, resulting in increased throughput. Figure 6.3b shows the 2 extremes from Figure 6.3a with burst size of 3 and 64, where the FIFO depth is changed between 7 and 64. Here it is checked what kind of influence the FIFO depth has on the throughput, with these two burst size extremes. A couple of things can be observed when comparing the graph with burst size 3 to the graph with burst size 64. If the burst size is 3, the throughput will saturate at a far smaller FIFO depth, compared to the burst size of 64. The reason for this is that for a packet with a small size, a smaller



(a) Throughput w.r.t. FIFO depth

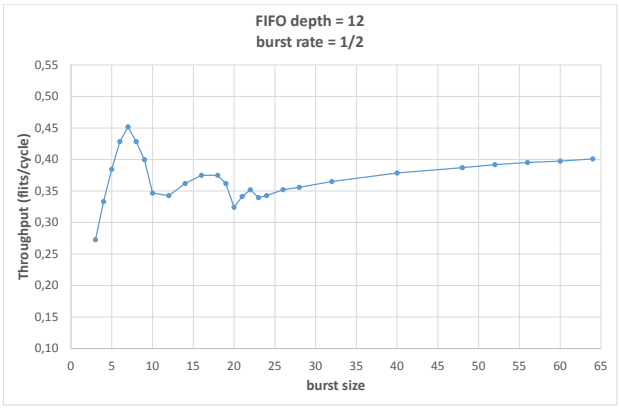


(b) Throughput w.r.t. FIFO depth (zoomed)

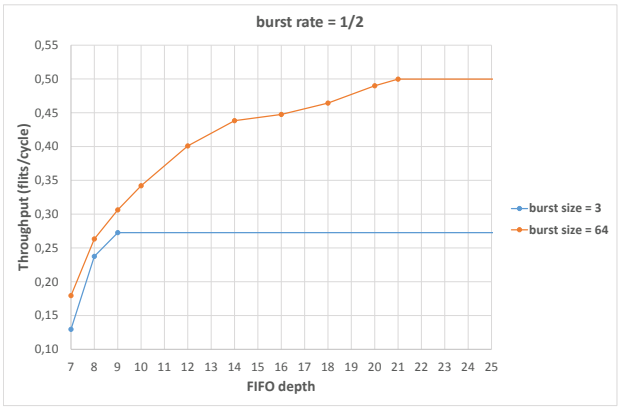
Figure 6.2: Throughput w.r.t. FIFO depth with flit injected on every clock cycle

FIFO size is needed to be able to fit into the FIFO. For a packet with a large size a larger FIFO size would be needed, in order to let the packet fit into the FIFO. Because of the fact that the throughput of a smaller packet saturates much earlier, this also results in a much lower maximum throughput. The reason for this is when an amount of flits is transported in smaller packets, the router has to route more packets compared to larger packets. And because a router needs time to route, more time is lost with the routing of smaller packets, compared to large packets.

Figure 6.4 shows the throughput when the burst rate is 1/8, with dependence on burst size or FIFO depth. In Figure 6.4a the case is shown where the FIFO depth is kept at 12 (same as S. Kumars findings), the burst rate is set on 1/8 and the burst size is changed to see how this influences the throughput. It is tested for a burst size between 3 and 64, and it can be observed that the throughput almost doesn't change



(a) Throughput w.r.t. burst size

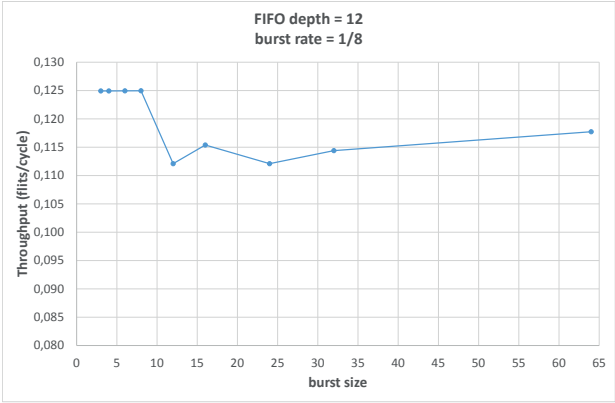


(b) Throughput w.r.t. FIFO depth

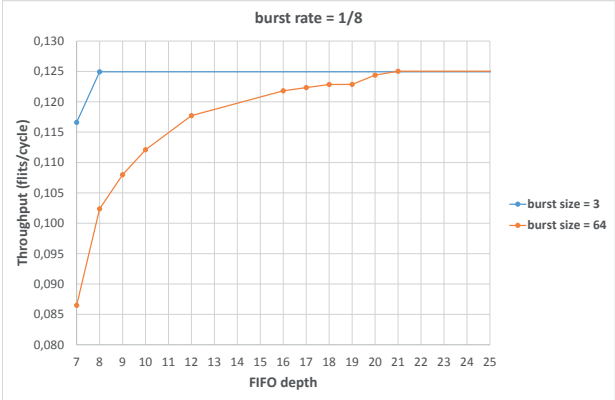
Figure 6.3: Throughput when burst rate is 1/2

at all. As in Figure 6.3a some oscillations can be seen, but because the throughput is so small, the difference is negligible. This has to do with the fact that the burst rate is 1/8, which causes less data to be transported. Figure 6.4b shows the 2 extremes from Figure 6.4a with burst size of 3 and 64, where the FIFO depth is changed between 7 and 64. As in Figure 6.3b, the smaller burst size saturates earlier, because of the same reasons. The big difference compared to Figure 6.3b is that the maximum throughput is the same for both the burst sizes. This has to do with the fact that the burst rate is 1/8, which means the FIFO is never fully occupied when using a burst size of 3.

In Figure 6.5 the clock cycle of one component is changed relative to the other two components in such a way, that it is 2, 4 or 8 times smaller. Here the influence of the FIFO depth can be tested. Figure 6.5a shows the case where the injector clock cycle is changed, Figure 6.5b shows the case where the clock cycle of the first router is changed,



(a) Throughput w.r.t. burst size



(b) Throughput w.r.t. FIFO depth

Figure 6.4: Throughput when burst rate is 1/8

and Figure 6.5c shows the case where the clock cycle of the second router is changed.

In Figure 6.5a it can be observed that the throughput is not determined by the injector. Even in the case where the injector runs at an 8 times faster frequency, the throughput for all the FIFO depths stays almost the same. In Figure 6.5b it can be observed the first router does exert influence on the throughput. If it's frequency increases, less FIFO depth is needed to achieve maximum throughput. The reason for this is, when the frequency of the first router is increased, the data from it's FIFO is read out faster. This results in less dependability on the FIFO depth, which eventually results in a maximum throughput at a smaller FIFO depth. In Figure 6.5c it can be observed that the throughput is not determined by the second router. The difference with the previous case is, that in this case the router gets it's data from a previous router, which has the same FIFO depth. The throughput is then determined by the

FIFO depth of the first router. It doesn't matter how fast the second router is, because it will stop sending flits when it's FIFO is full.

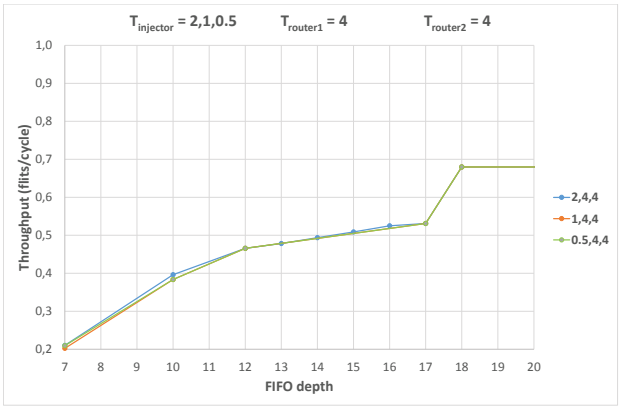
In Figure 6.6 the clock cycle of two components is changed relative to the other single component in a way, that they are 2, 4 or 8 times smaller. Here the influence of the FIFO depth can be tested. Figure 6.6a shows the case where the clock cycle of the injector and the first router is changed, Figure 6.6b shows the case where the clock cycle of the first and second router is changed, and Figure 6.6c show the case where the clock cycle of the injector and the second router is changed.

In Figure 6.6a a similar scenario as in Figure 6.5b can be observed, only here the injector also works at a higher frequency. This shouldn't change the throughput, because the second router remains the limiting factor. In Figure 6.6b an interesting case can be observed. Here both the routers are working on a fast frequency. This means that they should be able to process all the incoming flits from the injector, which are being sent with a slow frequency. The limited throughput of the routers doesn't play a substantial role here, because this shortcoming is compensated by their higher frequency. Because of this, the throughput is close to 1. In Figure 6.6c can be observed that if both the injector and the second router operate at a higher speed, the FIFO depth will exert influence on the throughput, the same way it does in Figure 6.5b.

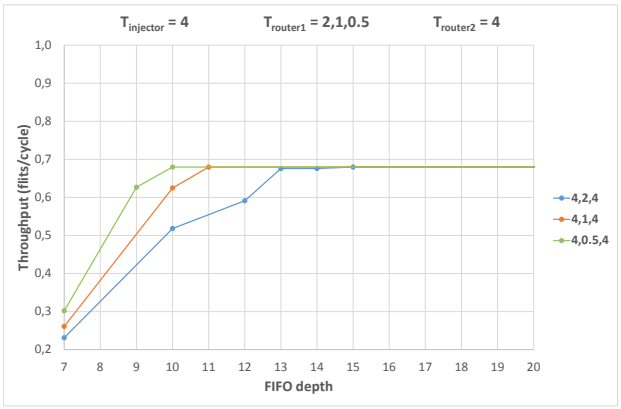
When comparing the throughputs for different FIFO depths, the FIFO depth has an influence only when 2 out of 3 components work on a faster frequency, or when the first router works at a faster frequency. The maximum throughput is determined by the slowest component, which can be observed in all the examples. Only when both the routers operate at a faster frequency, it's limited throughput is compensated, and almost all the flits from the injector will be processed.

Looking at all the test results, some conclusions can be drawn:

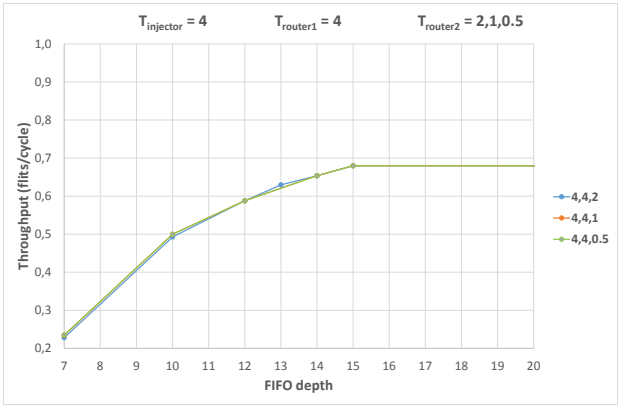
- For larger packet sizes (i.e. 17), the FIFO depth larger than that packet size has no influence on the throughput
- A FIFO size identical to the packet size is very beneficial for the throughput
- For smaller packet sizes, the FIFO depth has less influence on the throughput, than for the larger packets sizes
- For a larger throughput, it is better to use large packets. The drawback is that larger FIFO's are needed in order to be able to process these large packets fast enough
- For smaller burst rates, it is better to use smaller packet sizes, because in that case, the maximum throughput is achieved much faster compared to a large packet size
- The first router has the most influence on the throughput
- The best results are obtained when both the routers are operating at a larger frequency compared to the injector



(a) Injector is factor α faster then router 1 and router 2

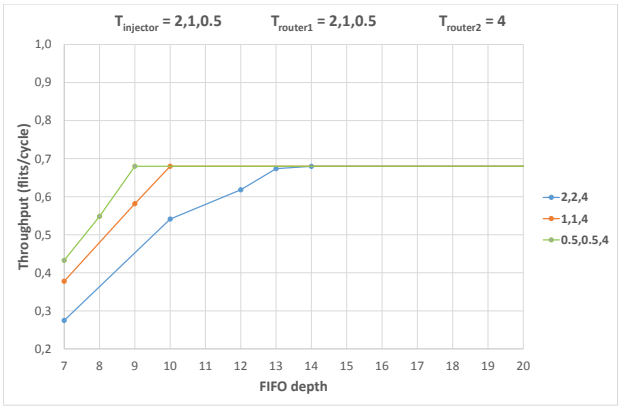


(b) Router 1 is factor α faster then injector and router 2

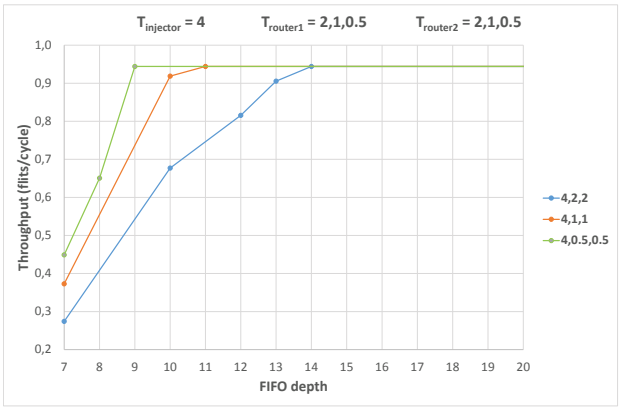


(c) Router 2 is factor α faster then injector and router 1

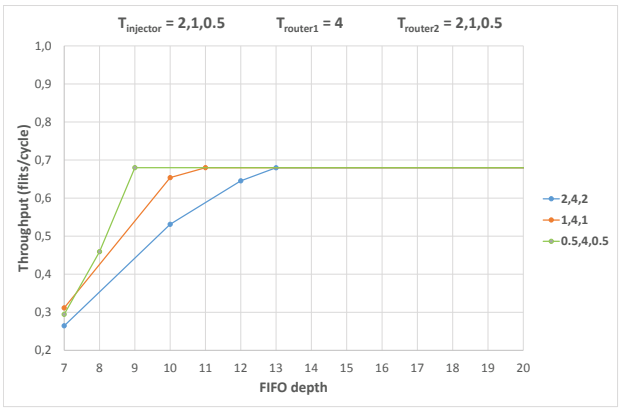
Figure 6.5: Throughput w.r.t. FIFO depth, where $\alpha = 2, 4, 8$



(a) Injector and router 1 are factor α faster then router 2



(b) Router 1 and router 2 are factor α faster then injector



(c) Injector and router 2 are factor α faster then router 1

Figure 6.6: Throughput w.r.t. FIFO depth, where $\alpha = 2, 4, 8$

6.1.3 Latency

In this experiment, the latency is tested for a burst size of 17, a burst rate of 1 and for FIFO depths between 7 and 64. This is done in case where all the components operate at the same frequency (see Figure 6.7a), one of the components is operating at a two times lower frequency (see Figure 6.7b) and two of the components are operating at a two times lower frequency (see Figure 6.7c). In case the frequency is equal for all components, the latency is given in cycles per flit. In all other cases the latency is given in absolute time [ns] per flit.

Figure 6.7a shows that the latency falls at the beginning, but after that keeps on rising, with a small decrease at the FIFO depth of 18. The router architecture by S. Kumar is designed in such a way, that after injecting a flit into a router, it goes in a FIFO and dependent on the flow control signal from the downstream router, the flit has to wait in the FIFO, resulting in an increased latency. If the FIFO is larger, the flit stays longer in the FIFO, resulting in a larger latency. Figure 6.7b shows that if the injector is operating at a lower frequency, the latency will not go up. The reason for this is that the flits are injected at a lower frequency, and processed by the two routers at a higher frequency. There is no stalling of flits in the routers, and that is the reason why the latency will not go up. The worst case is when the second router is operating at a lower frequency. In that case the latency is increasing the most. The reason for this is that the flits are injected at a higher frequency, and in the end they are processed by the second router at a lower frequency. In this case the FIFO will get full more often, which will result in an increased latency. Figure 6.7c shows that the lowest latency increase is obtained in the case the injector and the first router, or both of the routers are operating at a lower frequency. The worst case is obtained when the injector and the second router are operating at a lower frequency.

Looking at all the test results, some conclusions can be drawn:

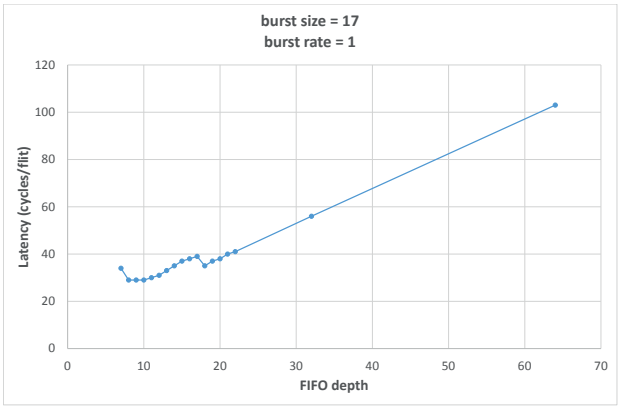
- The latency will increase for larger FIFO depth
- The latency will increase the most in the case when the second router is operating at a lower frequency
- The worst case is when both the injector and the second router operate at a lower frequency

6.2 1 router after P&R

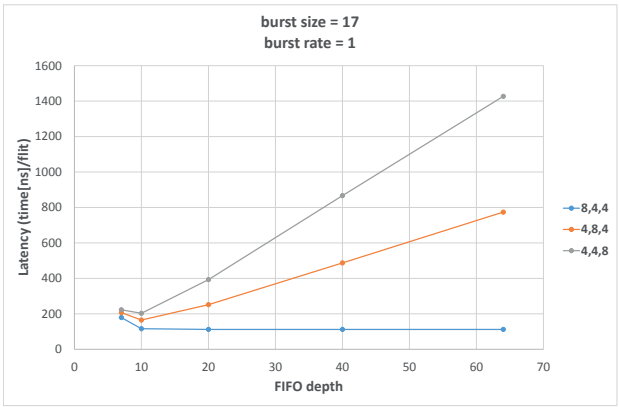
In this section, tests are performed on a placed and routed design of a single asynchronous 3D router with serialized vertical connections. Here different timing results will be given as well as area results.

6.2.1 Timing

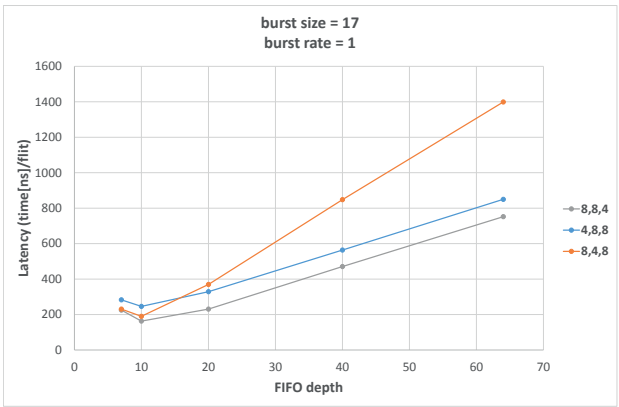
This router is simulated with a different *Serialization Ratio (R)* to determine what influence it has on the timing. The R which are tested are 1,2,4,5,8,10,20 and 40. For every R, it is determined at which fast clock cycle and which slow clock cycle this



(a) All clocks are the same



(b) One of the clocks is 2 times slower



(c) Two of the clocks are 2 times slower

Figure 6.7: Latency w.r.t. FIFO depth

design will have a positive slack. This is an indication that the design should be able to operate without any problems. The results are shown in Figure 6.8.

I can be seen that both the clock cycles are around 4 ns for $R = 1$. This is because in this case the critical path is limited by the slow clock cycle, which is also around 4 ns. If $R = 1$, then the fast clock cycle is also 4 ns. The same case can be described when $R = 2$, in which case the slow clock cycle is 4 ns, and the fast clock cycle is 2 ns. From $R = 4$ a trend can be observed, where the fast clock cycle stays around 1 ns, and the slow clock cycle is R multiplied by the fast clock cycle. This can be explained by the fact that the critical path of the fast clock is limited at around 1 ns, which means that the slow clock cycle is R times larger then the fast clock cycle.

It can be also observed that the critical path for the fast clock is not always 1 ns. This is because for all the experiments, the design is placed and routed again, with different clock trees and timing, which causes the small difference in the timing results.

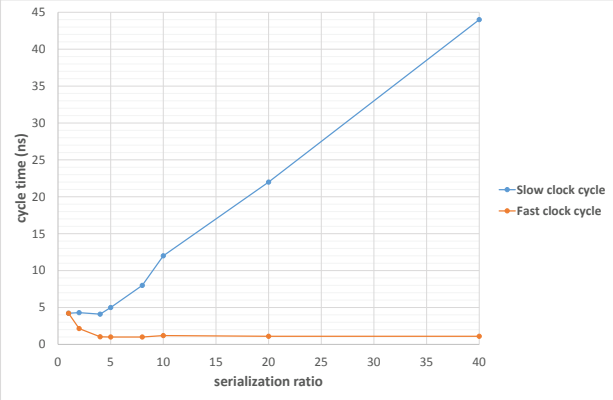


Figure 6.8: Cycle time w.r.t. serialization ratio

Looking at Figure 6.8 it can be concluded that the best R is 4. This results in the fastest fast clock, and fastest slow clock.

6.2.2 Asynchronous operation

This is a small section about the asynchronous operation of the router. The asynchronous operation means the data is written into the FIFO at one clock frequency and read out at another clock frequency. These clock frequencies have to be asynchronous to each other. For the tested clock cycle times in Table 6.1, the router is operating properly. This means that all the packets which are injected into the local port, are routed to the valid output port.

In Table 6.1, where T_W is the write clock cycle time and T_R is the read clock cycle time, it is shown how big the frequency difference can be between the two clocks, when the write clock is faster or when the read clock is faster. It is also shown for the two clocks when the phase between them goes up and down. This way it is tested if the router still operates when the two clock edges are close to each other. It can be

Table 6.1: Tested clock cycle time at which the router is operating properly

T_W [ns]	T_R [ns]	description
4	4	Write and read frequency are the same
4	8	Write frequency is 2x faster than the read frequency
4	16	Write frequency is 4x faster than the read frequency
4	32	Write frequency is 8x faster than the read frequency
4	64	Write frequency is 16x faster than the read frequency
8	4	Write frequency is 4x lower than the read frequency
16	4	Write frequency is 4x lower than the read frequency
32	4	Write frequency is 8x lower than the read frequency
4.12	4.16	Two fast frequencies where the phase goes up and down
4.16	4.12	Two fast frequencies where the phase goes up and down
128.12	128.16	Two slow frequencies where the phase goes up and down
128.16	128.12	Two slow frequencies where the phase goes up and down

concluded that it is operating properly. The only problem lies when the write frequency is lower than the read frequency. This has to do with the initialization of the memory after the reset, which is explained in Section 5.2.1.2. If the write frequency is too slow compared to the read frequency, memory initialization problems can arise. From the test results it can be seen that it still operates correctly when the write frequency is up to 8 times lower than the read frequency. When the frequency difference becomes larger, the memory initialization problems will arise. Due to time constraints, this particular problem is not solved.

6.2.3 Area

The area of the 3D router block is dependent on the components used. A large area footprint is taken by the Faraday memory block and the TSV array. When changing the memory or serialization factor, the area taken by them is also automatically changed. Figure 6.9 shows the floorplan used for the final design for 1 3D router with serializer/deserializer and synchronizers. Here the memory is shown for word depths of 16, 32 and 64. It can be observed that the memory footprint doesn't change a lot. This is also shown in Figure 6.11. This has to do with the fact that in this memory architecture, at these sizes, the area is occupied primarily by control logic instead of memory cells. Also the TSV area is shown for all the serialization ratios (between 1 and 40). The largest area footprint is for $R = 1$, and the smallest for $R = 40$.

The TSV array is made in such a way, that is should be as square as possible, which is explained in Section 2.1.4.3. This means that if possible, the number of rows should equal the number of columns. If not possible, then one of the two can be one larger than the other. When making this kind of square arrays, it can't be ensured that all the area in the square is filled with TSVs. There will also be some empty spots. Table 6.2 shows the number of TSVs needed with different serialization ratios, with the number of rows and columns needed for the array, and how many empty spots there are left.

The number of needed TSVs for different serialization ratios can be calculated by

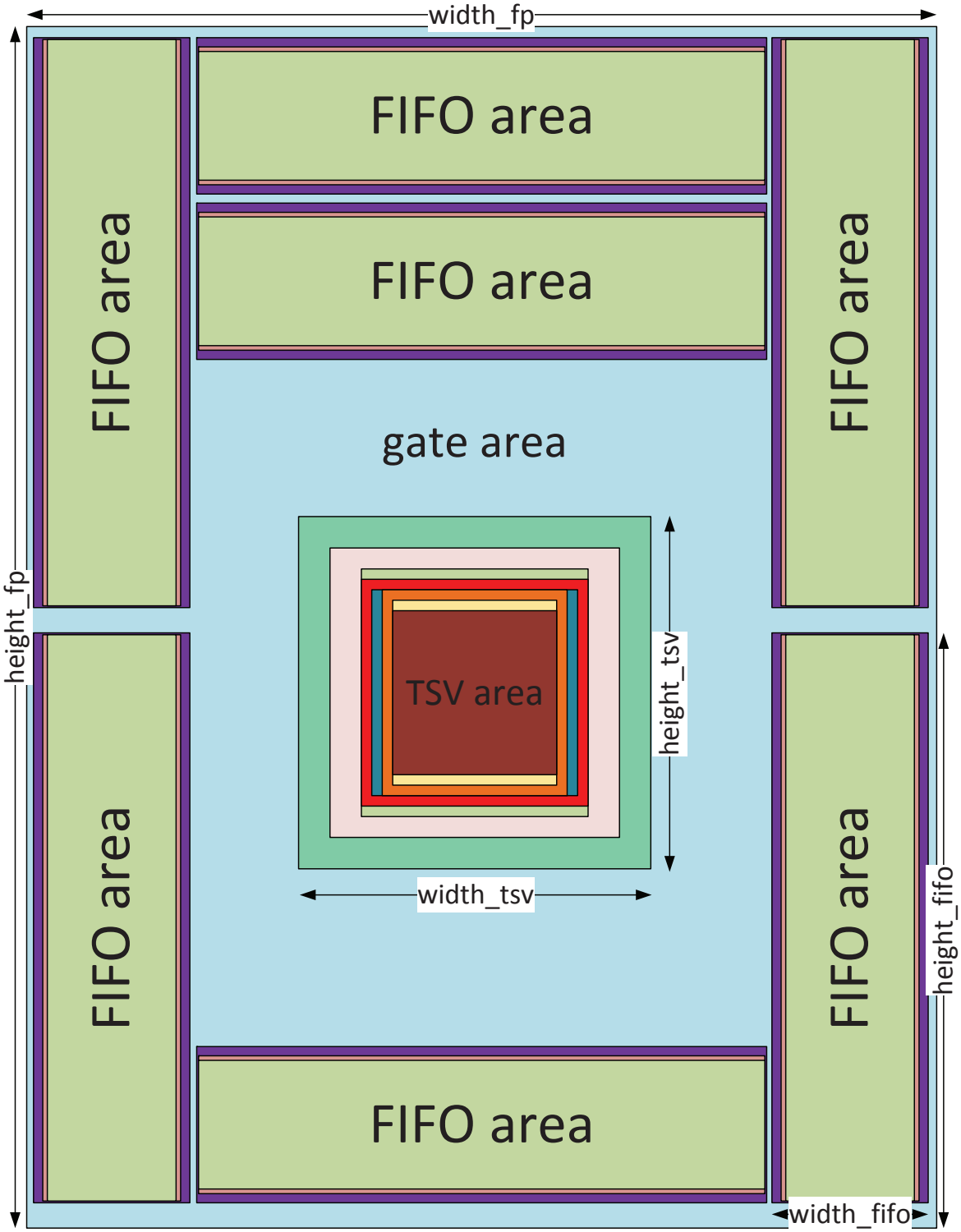


Figure 6.9: Floorplan of the 3D router block with different sizes of memory and TSV array w.r.t. serialization ratios

(6.1).

$$\text{number of TSVs needed} = 12 + 4 \times (40/R) \tag{6.1}$$

Table 6.2: TSV array with different serialization ratios

Serialization ratio	TSVs	Columns	Rows	Empty spots
1	160	13	13	9
2	92	10	10	8
4	52	7	8	4
5	44	7	7	5
8	32	6	6	4
10	28	5	6	2
20	20	4	5	0
40	16	4	4	0

Figure 6.10 shows the total TSV footprint (with KOZ), and gate footprint for different serialization ratios. It can be observed that the TSV footprint is reduced the most with $R = 2$ and 4, after which the reduction has less impact on the area. The gate area stays relatively the same.

The used chip area is determined primarily by the memory blocks. Because of performance issues, it is preferable to put these blocks on the I/O sides, because the I/O pins are connected to the memory blocks. This is shown in Figure 6.9. In Figure 6.9 and Figure 6.11 it can be observed how the memory area changes for different word depths. The difference between 16 and 64 bits is around 22%. But because of the placement of the memory, it has a small influence on the total block dimensions. For 32 word depth instead of 16, the block size can stay the same. Only for the increase to a 64 word depth, the size is increased by 1,5%.

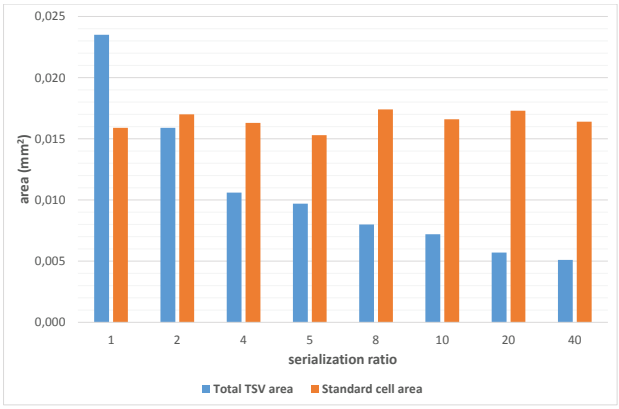


Figure 6.10: Total TSV and gate area w.r.t. serialization ratio

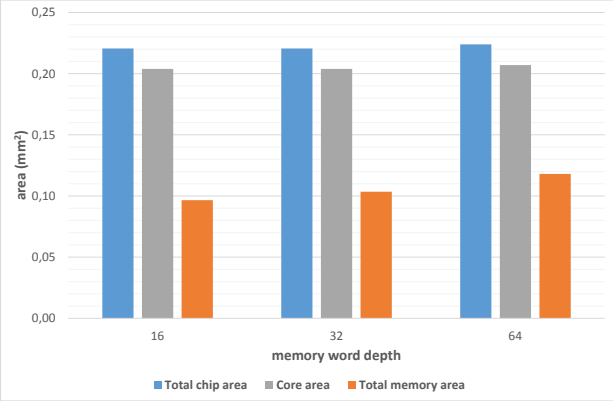


Figure 6.11: Total memory, chip and core area w.r.t. memory word depth

Looking at these results, it can be concluded that $R = 4$ is the best value for the area as well. It can also be seen that most of the area is occupied by memory blocks. A possibility to try out would be to make the memory blocks out of the Nangate Standard Cells. Maybe this way the memory area would decrease. Due to time constraints, this is not implemented.

7 Conclusion

This chapter summarizes this thesis by highlighting the most important aspects of the physical design of the 3D router, where information is given for all the important design steps, and achieved goals. Additionally, areas which could be further explored will be highlighted so future engineers can start working on them.

7.1 Summary

A solution has been given for various problems of chip design in the form of a 3D NoC, which solves the communication and area problems. The emphasis of this work has been the design of the main block of a 3D NoC, which is a 3D router. From the RTL level of design to a fully placed & routed design, all the design steps and problems are described.

First some background information is given in Chapter 2, where different aspects like IC design methodology, Standard Cell Library, Memory, Through Silicon Via and 3D implementation are discussed, to see what to use where, and how to implement them. For the standard cells, the Nangate Open Cell Library was chosen. The reason for this was that is was quite small (45 nm) and was ready for 3D implementation. For the memory, 55 nm Faraday memory is used, because this was the smallest available memory. Various TSV technology options are discussed, where it was shown that the TSV technology complementing the Nangate Standard Cells, had a diameter of 6 μm. In the end of the chapter a complete work flow is described for designing the chip.

In Chapter 3 the basic building block of a 3D NoC, the 3D router is discussed. First the architecture is determined, which is from S. Kumar, after which it is explained. The implementation of the memory blocks is explained, with the emphasis on making them usable in the design. This was needed because the Faraday memory was made in 55 nm, where the Standard Cell Library was made in 45 nm. Some physical information had to be changed, which was done with LEF files. Also some small adjustments were made in the Liberty file, which was needed for timing. The complete design and implementation of a TSV is described. For the geometry of the TSV, a square is used with sides of 4 μm. This was the easiest to implement and sufficient for it's purposes. The size is chosen based on the work of R. Jagtap, where it is used for some simulations, and it was close to the size of the TSV from the PDK. Using the values for the parasitic components for a TSV, from R. Jagtap [8], some simulations are done to determine the output transition time and propagation delay for TSVs w.r.t. input transition time and output capacitance. The result was that the output transition time was almost equal to the input transition time, and that the propagation delay was negligible. These results were used in the Liberty file, describing the timing for a TSV. Next to this, also a LEF file had to be described, where all the geometrical information is stored. For simulation

purposes, a simple RTL model had to be made in VHDL.

In Chapter 4 the serializer and deserializer are discussed. The full details of the design are given, together with the operation. Some aspects are highlighted, like timing constraints that needed to be obeyed in order to let the design operate in various conditions.

Chapter 5 describes the asynchronous working of the router, and how to implement it with the help of of synchronizers and gray en-/decoders.

In Chapter 6 some simulations are done. One is an RTL simulation with two routers connected to each other. Here the throughput is tested for different scenario's. It can be seen that the FIFO depth has influence up to a value of 18, after which the throughput stays the same at a maximum of 0.67, which is determined by the router architecture. Different burst sizes are tested. It can be seen that the throughput gets larger with larger burst sizes. The only drawback is that a larger FIFO is needed to take full advantage of the throughput. Also the FIFO depth has less influence on the throughput when the burst size is small, compared to large burst sizes. For smaller burst rates, it is better to use smaller burst sizes, because in that case, the maximum throughput is achieved much faster compared to a large packet size. Some tests were also conducted to see how an increase in frequency for 1 component has influence on the total throughput. It can be seen that the maximum throughput stays the same, but the FIFO depth needed to achieve it, will change. This is the case when the frequency of the routers is changed, and not of the injector. The tests where the frequency of 2 of the components is changed, support the above statement. One of the routers should operate at a higher frequency, in order to achieve the maximum throughput with smaller FIFO depths. The case where both routers are operating at a fast frequency is the most interesting. Here the throughput is almost 1, which means on every clock cycle of the injector, one flit is sent through the the network. This does make sense, because if the two routers are working at higher frequencies, they can process far more data then they receive, and in this way, compensate the limiting throughput of 0.67, imposed by the routers' architecture. The latency is also tested for different FIFO depths, for the case where all the components are operating at the same clock frequency, when one of the components is operating at a lower clock frequency or when two of the components operate at a lower clock frequency. For the case when the clock frequency is the same for all the components, it can be observed that the latency will increase with the FIFO depth. When testing with different clock frequencies, it can be observed that the latency will increase the most when the second router, or the injector and the second router are operating at a lower clock frequency, compared to the other component(s).

The other simulation is a placed & routed design of one router, including serializer/deserializer and synchronizers. Here the timing and area are discussed. It is observed that the critical path of the slow clock lies around 4 ns, and of the fast clock around 1 ns. This means that the most efficient serialization ratio is 4. Together with the fact that the TSV array area decreases the most with serialization ratios of 2 and 4, the value 4 can be seen as the most efficient. The gate area stays the same with different serialization ratios. Furthermore the memory size doesn't increase substantially when increasing the word depth from 16 to 64. This has to do with the fact that in this memory architecture, at these memory sizes, the area is primarily occupied by control

logic instead of memory cells. Some test results are given showing that the router also works with two asynchronous clock frequencies.

7.2 Future work

Due to timing constraints, some aspects are not examined thoroughly enough, or not at all. These could be interesting for future engineers who wish to continue on this work. The areas where future work could be done are the following:

1. For this project, no real 3D design has been made because no 3D EDA tools were available. With small adjustments, the 3D router could be implemented with real 3D EDA tools.
2. The emphasis of this project was the development and implementation of 1 3D router. Maybe a real 3D NoC could be made and tested.
3. The used Nangate Standard Cell Library has no industrial grade performance. This design could be synthesized with industrial grade Standard Cell Libraries.
4. The used Faraday memory seems to take up a lot of area. Maybe other more area efficient memories could be used, or the memory could be synthesized with ordinary standard cells. A comparison could be made with the different implementations.
5. This thesis did not look at the power dissipation at all. Especially in today's world where low-power becomes the norm, it would be interesting to see how much power this design dissipates, and if it can be reduced.

TSV timing values for liberty file

A

In Table A.1 all the output transitions are given with respect to input transitions and output capacitance, where in Table A.2 all the propagation delays are given with respect to input transitions and output capacitance. The input and output transition times are calculated from 30% to 70%, and the propagation delays are calculated from 50% input to 50% output.

Table A.1: Output transition time w.r.t. input transition time and output capacitance

Output transition time (s)	**Input transition time (ns)**						
Output capacitance (ff)	0.002	0.011	0.042	0.102	0.196	0.327	0.500
0.365	2.36p	11.26p	42.68p	102.70p	196.19p	327.37p	499.99p
1.897	2.30p	11.26p	42.68p	102.70p	196.19p	327.37p	499.99p
3.795	2.35p	11.27p	42.68p	102.69p	196.19p	327.37p	499.99p
7.591	2.43p	11.28p	42.69p	102.69p	196.19p	327.37p	499.99p
15.182	2.64p	11.20p	42.70p	102.69p	196.19p	327.37p	499.99p
30.365	2.77p	11.30p	42.69p	102.69p	196.19p	327.37p	499.99p
60.730	1.41p	11.42p	42.68p	102.70p	196.19p	327.37p	500.00p

Table A.2: Propagation delay w.r.t. input transition time and output capacitance

Propagation delay (s)	**Input transition time (ns)**						
Output capacitance (ff)	0.002	0.011	0.042	0.102	0.196	0.327	0.500
0.365	78.34f	3.86f	2.87f	3.12f	3.02f	2.75f	2.69f
1.897	140.41f	4.79f	2.98f	3.50f	3.43f	3.13f	3.05f
3.795	205.43f	9.28f	3.59f	3.50f	3.96f	3.61f	3.52f
7.591	294.38f	-8.30f	4.02f	4.36f	4.95f	4.58f	4.42f
15.182	-8.17f	-15.26f	6.65f	6.06f	6.64f	6.52f	6.23f
30.365	-465.15f	43.28f	10.51f	11.48f	9.93f	9.95f	9.88f
60.730	-248.93f	-34.35f	1.52f	14.62f	16.05f	16.84f	17.17f

3D router diagram

B

Figure B.1 shows a 3D router diagram where only the LOCAL and UP input/output ports are shown, because all other ports are the same.

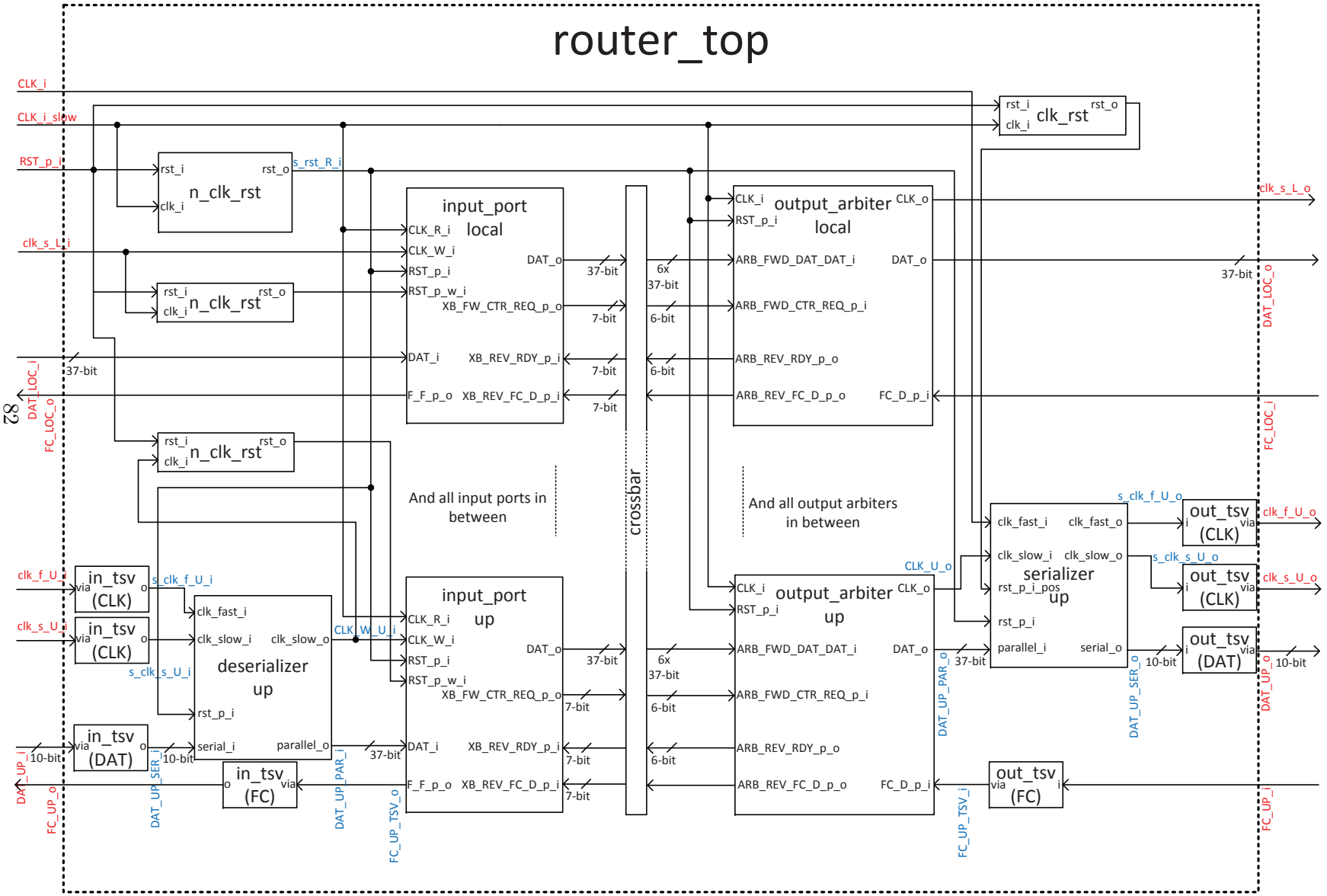


Figure B.1: Complete 3D router diagram with LOCAL and UP input/output port

Bibliography

[1] "FreePDK3D45: Metal Layers," www.eda.ncsu.edu/wiki/FreePDK3D45:Metal_Layers, Retrieved 29 April, 2013.

[2] J. Hoekstra, "Introduction to Nanoelectronics: Single-Electron Electronic Circuits," 2010.

[3] K. Tatas, K. Siozios, D. Soudris, and A.l Jantsch, *Designing 2D and 3D Network-on-Chip Architectures*, Springer Science+Business Media New York, 2014, ISBN: 9781461442738.

[4] J.M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, Pearson Education, 2003, ISBN: 0131207644.

[5] M. Schoeberl, F. Brandner, J. Sparso, and E. Kasapaki, "A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems," .

[6] S.S. Kumar, "TMFab: A Transactional Memory Fabric for Chip Multiprocessors," 2010.

[7] C.E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO design," *SNUG*, 2002.

[8] R.S. Jagtap, "A Methodology for Early Exploration of TSV Interconnects in 3D Stacked ICs," Tech. Rep., Delft University of Technology, September 2011.

[9] "xor2 standard cell family," www.vlsitechnology.org/html/cells/vgalib013/xor2.html, Retrieved 29 April, 2013.

[10] "Library Overview," www.si2.org/openeda.si2.org/projects/nangatelib, Retrieved 29 April, 2013.

[11] NCSU, www.eda.ncsu.edu/wiki/FreePDK45:Manual, *FreePDK45 Manual*, product version 1.4 edition, July 2011.

[12] F. Clermidy, F. Darve, D. Dutoit, W. Lafi, and P. Vivet, "3D Embedded Multi-core: Some Perspectives," *CEA-LETI*, November 2011.

[13] L. Cadix, M. Rousseau, C. Fuchs, P. Leduc, A. Thuaire, R. El Farhane, H. Chaabouni, R. Anciant, J.-L. Huguenin, P. Coudrain, A. Farcy, C. Bermond, N. Sillon, Fléchet, and P. Ancey, "Integration and Frequency dependent electrical modeling of Through Silicon Vias (TSV) for high density 3DICs," 2010.

[14] C. Liu and S.K. Lim, "A Design Tradeoff Study with Monolithic 3D Integration," in 13^{th} *Int'l Symposium on Quality Electronic Design*. IEEE, 2012, pp. 529–536.

[15] M. Jung, J. Mitra, D.Z. Pan, and S.K. Lim, "TSV Stress-Aware Full-Chip Mechanical Reliability Analysis and Optimization for 3-D IC," in *Transactions on computer-aided design of integrated circuits and systems*. IEEE, August 2012, vol. 31, pp. 1194–1207.

[16] T. Frank, S. Moreau, C. Chappaz, L. Arnaud, P. Leduc, A. Thuaire, and L. Anghel, "Electromigration Behavior of 3D-IC TSV Interconnects," IEEE, 2012, pp. 326–330.

[17] A.-C. Hsieh and T.T. Hwang, "TSV Redundancy: Architecture and Design Issues in 3-D IC," *Transactions On Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 4, pp. 711–722, April 2012.

[18] I. Loi, S. Mitra, T.H. Lee, S. Fujita, and L. Benini, "A Low-overhead Fault Tolerance Scheme for TSV-based 3D Network on Chip Links," pp. 598–602, 2008.

[19] E. Beyne, "Electrical, Thermal and Mechanical Impact of 3D TSV and 3D Stacking Technology on Advanced CMOS Devices - Technology Directions," .

[20] Y. Civale, S. Armini, H. Philipsen, A. Redolfi, D. Velenis, K. Croes, N. Heylen, Z. El-Mekki, K. Vandermissen, G. Beyer, B. Swinnen, and E. Beyne, "Enhanced Barrier Seed Metallization for Integration of High-Density High Aspect-Ratio Copper-Filled 3D Through-Silicon Via Interconnects," pp. 822–826, 2012.

[21] A. Mercha, G. van der Plas, V. Moroz, I. de Wolf, P. Asimakopoulos, N. Minas, S. Domae, D. Perry, M. Choi, A. Redolfi, C. Okoro, Y. Yang, J. van Olmen, S. Thangaraju, D. Sabuncuoglu Tezcan, P. Soussan, J.H. Cho, A. Yakovlev, P. Marchal, Y. Travaly, E. Beyne, S. Biesemans, and B. Swinnen, "Comprehensive Analysis of the Impact of Single and Arrays of Through Silicon Vias Induced Stress on High-k / Metal Gate CMOS Performance," pp. 2.2.1–2.2.4, 2010.

[22] M. Jung, X. Liu, S.K. Sitaraman, D.Z. Pan, and S.K. Lim, "Full-Chip Through-Silicon-Via Interfacial Crack Analysis and Optimization for 3D IC," pp. 563–570, 2011.

[23] E. Beyne, "3D Interconnection and Packaging: Impending Reality or Still a Dream?," *International Solid-State Circuits Conference*, 2004, ISBN: 078038267604.

[24] NCSU, www.eda.ncsu.edu/wiki/FreePDK3D45:Manual, *FreePDK3D45 Manual*, product version 1.1 edition, July 2011.

[25] J. Bhasker, *THE EXCHANGE FORMAT HANDBOOK: A DEF,LEF,PDEF,SDF,SPEF&VCD PRIMER*, Star Galaxy Publishing, 2006, ISBN: 0965039137.

[26] I. Savidis and E.G. Friedman, "Closed-Form Expressions of 3-D Via Resistance, Inductance, and Capacitance," *Transactions On Electronic Devices*, vol. 56, no. 9, pp. 1873–1881, September 2009.

[27] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer," *International Symposium on Asynchronous Circuits and Systems*, 2003.

[28] "Clock Domain Crossing," filebox.ece.vt.edu/~athanas/4514/ledadoc/html/pol_cdc.html Retrieved 29 April, 2013.