Desynchronization of digital circuits

Rasmus Madsen

Kongens Lyngby 2011
IMM-M.Sc-2011-32

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

IMM-M.Sc: ISSN 0909-3192, ISBN 32

Abstract

In theory asynchronous circuits hold some great advantages over synchronous circuits, they are more robust towards variations in the environment such as temperature changes and voltage drops. At the same time asynchronous circuits can be compared to fine grained clock gating of a synchronous circuits, which if the circuit has idle time could save power. Finally asynchronous circuits does not have a finite clock cycle it consists of multiple local clocks generated by handshake controls, this should introduce a reduction in current spikes and EMI noise.

The use of asynchronous circuits today is limited to small scale prototyping and research experiments, the reason is that the computer aided design tools does not support the design flow for asynchronous design. Also designing asynchronous circuits is not so straight forward as designing synchronous ones, and especially debugging can be some what of a challenge.

This Thesis focuses on developing a method of desynchronization, to change a synchronous circuit into the asynchronous equivalent only by removing the clock, and by substitution of flipflops with latches. The first task is to implement some basic components in VHDL and create behavioral versions, the second task is to create synthesizeable versions of these components. Third task is to test on some examples and to establish a design flow for the synthesis and test of desynchronous circuits.

Preface

This thesis was carried out at the institute of Informatics and Mathematical Modeling of Technical University of Denmark as a requirement for obtaining the M.Sc. in engineering. the thesis is credited 30 ECTS points

The work was carried out from October 2010 to May 2011 under the supervision of docent Jens Sparsø.

I would like to thank my supervisor Jens Sparsø for great support and guidance throughput the project. Also i would like to thank Alberto Nannarelli and Massimo Petricca for their invaluable help on the synopsys packages

Finally I would like to thank family and close friends for their help and support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Project Motivation

Most circuits today are synchronous and with the scaling of the chips into the sub micron, it becomes increasingly difficult to cope with circuit variations such as clock-skew, voltage drops and temperature variations. [3, 2, 4] This is because all variations have to be accounted for in the design phase. One of the methods currently used is the SSTA (statically static timing analysis [1]) where as many variation parameters as possible are included to get a worst case estimation. The problem with this solution is that once the design is manufactured, it cannot adapt to changes and often results in including huge margins in the design, in terms of timing. Another solution is to use elastic or adaptive design [3]. Elastic or adaptive circuits are tolerant towards variations in the timing of the circuit due to temperature changes, meaning even if the speed of a part of the circuit is reduced, the behavior will still be correct.

A perfect example of an elastic circuit type is Asynchronous circuits. If asynchronous circuits are robust towards the variations previously mentioned, it might seem strange that almost no company implements this design strategy into their commercial design. The reason for this could be that Asynchronous circuits are very different and more difficult to design compared to synchronous

circuits, and the fact that there are no CAD-tools (Computer Aided Design tools) that support the synthesis and design flow of these makes designing a tedious and time consuming task.

Desynchronization is the technique of taking a synchronous design or specification and turning it into an asynchronous equivalent, by replacing the clock tree and registers with latches and latch controllers that use a handshake protocol to create local timing. The purpose of this thesis is to investigate the different methods of desynchronization and to evaluate the possibility of handling these using common CAD tools, by proposing a tool and design flow.

1.2 Aims of this thesis

The Goals of this thesis are

- to show the theory behind Desynchronization on a simple circuit using boolean equations and simulate and check behavioral
- Establish a tool flow
- Establish Design flow
- Test on real Examples

1.3 Thesis Overview

The Construction of the remainder of this thesis is chapter: 2 gives an introduction to asynchronous circuit design, the chapter presents the needed background to be able to desynchronize synchronous circuits. The last part of the chapter presents the tools used in the tool flow.

Chapter 3 gives a description of the basic components used in desynchronization, and it also describes the behavioral implementation in VHDL.The second part of the chapter describes the desynchronization design flow, from synchronous specification through desynchronization, synthesis and floorplanning and finally verifying the design.

Chapters 4,5,6 present 3 different examples of desynchronization, the implementation and the test results.

Finally chapters 7 & 8 contains the discussion and conclusion respectively.

Chapter 2

Desynchronization

The following chapter will give and introduction to the asynchronous circuit design methodology, and in detail explain the differences between synchronous and asynchronous design. Also the basic concepts like handshake, handshake protocols, data validity are explained. The last part of this chapter gives a short presentation of the problem of the EDA tools used today.

2.1 Introduction

Most digital circuits today have a globally distributed clock, which dictates the time in a discrete manner. These are called synchronous systems. The sequence of events is easy to understand since all events happen at the same time, namely every time the clock ticks. This means that the designer knows at exactly what point in time the data should be valid for all registers. The alternative to a synchronous system is an asynchronous system. In an asynchronous system, the clock is substituted with a set of handshake signals, that indicate when new data is available and when this data has been stored. One can say that the system is locally timed. Local clock signals are generated by the handshake controls. The asynchronous designs are a lot more complex to understand, since events will happen at what appears random times. There is no obvious sequence

to follow and data is valid at different points in time. In theory, asynchronous designs have some great advantages over synchronous ones.

- Low Latency - the speed is determined by local delays and not by the slowest part of the design.
- Low Power consumption - in asynchronous design global idling is implied, which means that only components that are needed is active, the rest is in an idle state not consuming power. This can be compared to fine-grained clock gating in a synchronous system.
- No clock distribution or clock skew problems - the clock is substituted by handshake protocols.
- More robust against voltage drops and temperature variations - The matched delay element will if routed properly be in the same area of the chip and experience the same temperature differences therefore behave in the same way as the corresponding combinatorial logic.
- Less sensitive to fabrication parameter variation
- Less Electromagnetic Interference (EMI) - since the system is locally timed the ticks of each "clock" happens at random points in time.
- Smaller current peaks, and smoother current consumption - the consumption is spread over time.

2.1.1 Asynchronous design

To understand desynchronization, one must first know the basics about asynchronous circuits. As stated in the introduction the asynchronous circuits does not have a globally distributed clock, but is timed by latch controllers linked by a handshake protocol. In handshake protocols there is a sender and a receiver, and both can be the initiator of a handshake sequence. When the initiator is ready it sends a request signal to the receiver telling that the next handshake sequence can begin. When the receiver has processed the request it sends an acknowledge signal telling that the required action has been completed(sending or storing data).

2.1.2 Handshake protocols

There are two main types of handshake protocols:

- bundled-data
- dual-rail data

The bundled-data has the request and acknowledge signals bundled together with the data, but as separate signals. The dual-rail protocol has the request and acknowledge signals incorporated in the data. This gives the most robust design in terms of delay insensitivity and parameter variation, but also holds the most complex implementation and area overhead.

Bundlet data protocol The bundled-data protocol can be split into two-phase and four-phase bundled data. The difference being in the four phase bundled data after each data transfer the request and acknowledge signals must return to zero, so this is a level sensitive signal where only the 1 has meaning. Every handshake must end with a return to zero period. This is costly in terms of time and energy. The two-phase bundled data has the request and acknowledge incorporated in the transitions, such that the transition 0 to 1 and 1 to 0 bears the same meaning. [6]. In theory the two-phase is faster and costs less energy, but is more complex to implement in reality. In the remainder of this thesis, only the four-phase handshake protocol will be used the reason for this that in [6] it is stated to be the one that resemble synchronous behavior the most, and is less complex than the alternatives. For a more detailed explanation of the handshake protocols please see [6].

In all types of handshake protocols we distinguish between push and pull channels usually marked by a little dot in the corner of the initiating controller see fig: 2.1. In the push channel case, latch-controller N sends a request to latch controller N+1 telling it that it has new data ready to be sent. When ready, the receiver stores the data in a latch and sends an acknowledge signal telling the sender that the data has been received. The sender then takes the request signal down, after that the receiver takes the acknowledge signal down, the handshake sequence is over, and the next one can begin. It is vital that the request does not arrive before the data is ready. The event at sender must be preserved at the receiver end, this is achieved using delay elements matched to the delay through the data path these are discussed in detail in 3. In the case of a pull channel, the N+1 controller sends a request signal saying that it is ready to receive new data. When new data is ready, the N_{th} sends and acknowledge signal along with the data. When the data is stored by the initiating controller, it pulls down the request signal, after which the sending controller pulls down the acknowledge signal, just as in the case of the push channel. The difference is the direction of

the request and acknowledge signals. In the case of a pull channel it is vital that the acknowledge signal does not arrive before the data is valid at the receiving end. To select a push channel over a pull channel is up to the designer and the application.

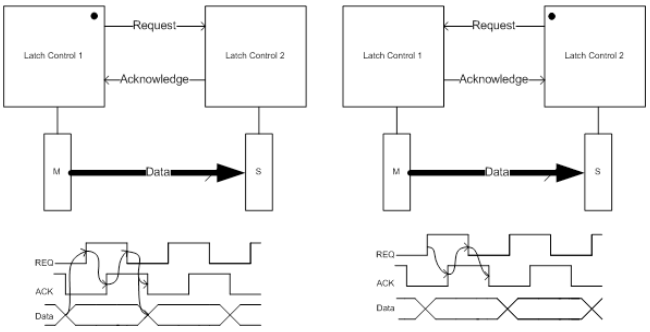


Figure 2.1: Left: Push Channel - Right: Pull Channel

2.1.3 Data validity

When using bundled data, it is important to define when data is valid on the receiving end. There are four different validity schemes for four phase bundled data [6]. Common for all of them is that they express the requirement set by the receiving end. In all of them, data should be valid some time before the request signal arrives, and some time after the acknowledge signal. This can be compared to the setup and hold constraints in synchronous design. The Choice of validity scheme affects the implementation of the handshake component in terms of area and speed, and therefore in some cases it can be advantageous to use a mix of the different schemes. The four schemes are early, broad, late and extended early, see fig: 2.2

Four data schemes for a push channel is listed below:

- In the case of the Data early scheme, the data is only valid from the request signal is received until the acknowledge signal is sent from the receiver.
- Extended early guarantees valid data from receiver sees the rising request signal until the request signal is pulled low again.
- With the broad scheme data is valid from rising request signal until the falling acknowledge signal event.

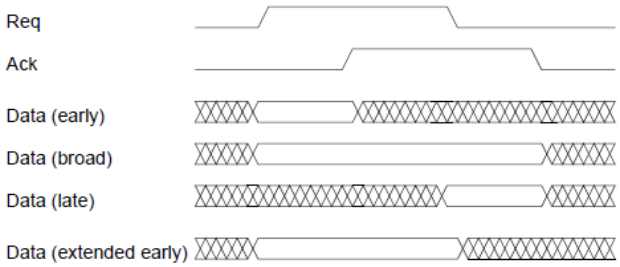


Figure 2.2: Validity schemes from [6]

- The final scheme is the data late, in which data is only valid from request falling event until the falling acknowledge event.

2.2 Desynchronization fundamentals

2.2.1 Desynchronization of a simple pipeline stage

When designing modern digital circuits, most of the time one strives for performance goals in terms of speed (latency and throughput), low power, area, and robustness. When it comes to speed, the latency of a circuit is determined by the critical path of the system (the slowest path). To increase throughput, circuits are often pipelined, and the slowest pipeline stage then determines the clock period. The clock must be adjusted so there is enough time to complete the calculation in the slowest stage. This naturally slows down faster stages which then have to wait for the slow one to complete before starting the next calculation. fig 2.3 show a synchronous pipeline, for all stages to be able to complete the clock must have a cycle time of 12ns. This gives a latency of 3*12ns = 36ns.

Asynchronous circuits do not have this drawback. Since every stage is locally timed the latency of a circuit is equal to the sum of the delays in each pipeline stage(29ns) fig 2.4. The deeper the pipeline, the greater the advantage. This advantage drops with the increase in amount of data. If the pipeline is busy 100 percent of the time, the faster stages will be stalled waiting for the slow stage to finish. Therefore there is only a gain in terms of latency up until a certain

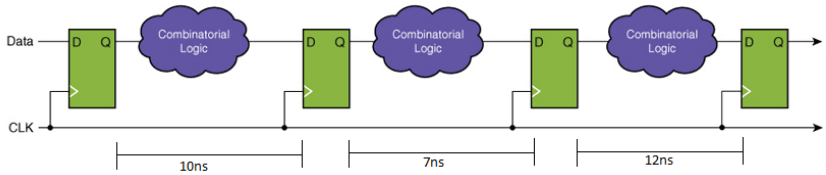


Figure 2.3: Synchronous Pipeline

occupation of the pipeline.

Low Power In the pipeline fig 2.3 all registers are clocked no matter if new data is present or not, this is very expensive in terms of power. A way to minimize this is to clock-gate parts of the circuit, to turn of parts of the circuit that are not used. This is implicit in asynchronous circuits since only the parts currently being used are active, the only power being dissipated in the idle part is due to leakage, this can be compared to a very fine grained clock-gating, and can result in a reduction of the overall power consumption. Again if the system is busy 100% of the time, Asynchronous circuits might be more power hungry due to the overhead in area (latch-controllers,forks and joins etc).
Desynchronization is a method to convert synchronous clocked gate logic into an asynchronous equivalent. By substituting flipflop registers with latches, and the clock-tree with a latch controlled handshake circuit leaving the combinatorial parts untouched, only the timing of the circuit has been modified, the datapath, and therefore the behavioral is the same.

Desynchronization is in theory straight forward, and is done in three steps

- Substitute all registers (flipflops) with a Master/Slave latch design
- Measure delay through every combinatorial path of the design for delay matching
- Implement latch controllers and delay elements in appropriate places

Clock skew With the introduction of the nanometer scale designs, distribution of the clock is increasingly difficult, the fact that the clock may arrive later in some areas of the design than others, pose a great challenge for the designers. This problem is not present in asynchronous design, since there is no clock!

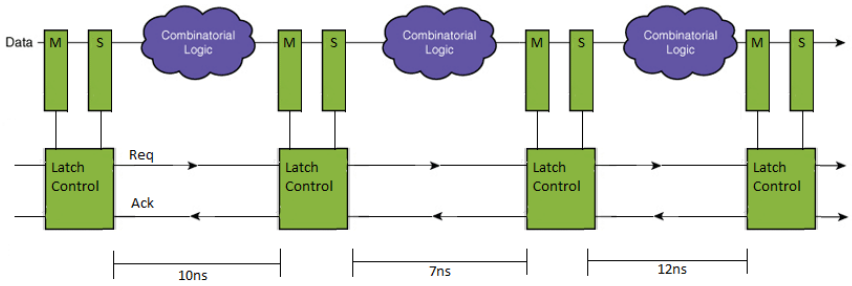


Figure 2.4: Asynchronous Pipeline

2.2.2 Granularity

One of the mentioned benefits of desynchronization is a decrease in power consumption, since this is directly linked to the switching activity in the circuit. And by removing the clock, the registers only switches when needed to. One of the drawbacks of desynchronization is the overhead that comes with implementing latch controllers, also two latches has a small area overhead compared to a flipflop, some times the synthesis library used includes a register with access to both latches inside, then the area is the same for the two latches. So more latch controllers obviously result in a bigger area overhead. This leads to the question of granularity.
How fine grained should the desynchronization be? As always there is no finite answer, it depends on the application, but some guidelines for a set of best practice are presented here:

- A separate controller should be used anywhere where data might arrive at different times.
- A separate controller should be used anywhere where combinatorial logic has more inputs, where only some is used in a given calculation. If there are eight inputs to some logic put only two is needed in a given calculation, it does not make sense to wait for all eight latches to fill up, the circuit should continue as soon as the need values are ready. The guidelines will be further explained through examples in 5
- In a combinatorial network receiving multiple inputs and always needing all inputs, the latches holding the input data for the network should be

controlled by the same controller. There is no point in having three controllers for three sets of latches if all three always switches at the same time. This would result in an unneeded overhead, also a join is needed to merge all request signals into one request for the stage after the combinatorial logic.

2.2.3 Methods of desynchronization

There are two obvious ways of desynchronizing a circuit. One is desynchronizing the VHDL code, following the steps described in 3.3. This method is intuitive and is probably the easiest to do when desynchronizing manually, because the hierarchial structure of VHDL makes it easy to navigate and find connections between components. This form of desynchronization, is done before synthesis. There is another way to desynchronize, it is possible to do it after synthesis by desynchronizing the synthesized netlist. Netlists are not difficult to read, but they are definitely not as easy and intuitive as the VHDL code, and for large designs it can be very difficult to keep track of nodes and wires. The netlist could be an excellent choice for an automatic desynchronization algorithm. This is beyond the scope of this thesis, the interested reader is encouraged to check out [4, 5] for further reading about this subject.

2.2.4 Pros and cons of desynchronization

In reality desynchronization although the idea is simple, it is not so. One of the main reasons for this is the lack of cad tools capable of handling the task. But there is also the question of data-dependency and delay matching, and that is why the use of asynchronous digital design is still limited to university research and small scale prototyping. Also the fact that designers have to completely rethink the way they design digital electronics, from clock ticks clearly indicating when data is valid to a design where the different parts of the circuit deliver valid data at random points in time. The fact that there is no global timing, indicating when data is ready also makes testing and debugging very difficult. when adding test stimuli to a circuit, the designer must make sure that the input data is synchronized the corresponding request and acknowledge signals. The gains of desynchronization have been presented in this chapter and it should be clear that, at least in theory, a desynchronized circuit holds some advantages over the synchronous equivalent.

2.3 CAD tools - Basics

This section is briefly commenting on the lack of EDA tools for asynchronous design, and introducing the tools used for the design flow.

2.3.1 Modern EDA tools

The EDA (Electronic Design Automation) tools today cannot handle asynchronous designs. The reason stated is, while having advantages and drawback none of the proposed methodologies can produce an asynchronous circuit with all the stated advantages [5] and therefore has not been adapted into any design tools. There are also no CAD tools available for asynchronous synthesis, which further complicate things, and forces designers to create own libraries or develop own tools. The Muller C element is not a part of any library, but this can be synthesized as a combination of simple gates.The routing of handshake signals poses a challenge to the tools. The complicated timing implications makes it complicated for the synthesis tools.

2.3.2 The tools used

The synthesis tool used in this thesis is Synopsys design compiler. The tool is not directly able to handle desynchronized designs. How this is done is explained in detail in 4, 4, 6 Synopsys design compiler takes the VHDL files and compiles them into a Verilog netlist that is then synthesized and floorplanned into a new verilog file that can be used for simulation of the design with actual delays etc. It also produce some reports of the circuit delay, node capacitances etc important when delay matching, and performance investigation. for simulation of the RTL, and Synthesized designs Modelsim is used Modelsim can handle mixedmode VHDL/Verilog files which makes simulating the synthesized Verilog netlists using the original VHDL test bench easy. To compare the desynchronized design with the original synchronous one. A Special matlab script has been developed. The script takes a VCD file (Value Change Dump) and a file containing node loads, and counts the switching activity.

this chapter introduced the basics of asynchronous circuits, and explained the theory behind desynchronization, finally a short introduction to the tools used for synthesis and test where presented, these are explained in detail in chapter 3, also the flow from VHDL to synthesized design is explained in detail.

Chapter 3

Basic components and design flow

The first part of this chapter will present the basic components of asynchronous systems, the functionality of each of them will be explained in details, and how and where they are used. The chapter will also present both behavioral models for easy functional testing and synthesizeable models for implementation. The second part will present the design flow of desynchronization from synchronous VHDL specification to synthesize and simulation of the desynchronized design using synopsis and modelsim.

3.1 Basic components

3.1.1 The Muller C element

To design asynchronous systems with correct behavior, one must take a look at when signals are required to be valid. In synchronous design the clock tick is used as an indicator of when all signals are valid. In between these ticks the signals may exhibit hazards. Hazards in this case are when signal level changes are not acknowledged by the system. In asynchronous design there is no click indicating that signals are valid and therefore signals must be valid at all times.

In chapter 2 the four phase bundled-data protocol was presented. Recalling the handshake sequence for a new sequence to begin both Request and Acknowledge signal must be 0 before a new handshake sequence can begin. At the same time the controller must hold the data until the next controller has received it, this pose a problem if we are limited to conventional logic gates. See 3.1 here 3 controllers in a pipeline are shown, the request signal is generated from the previous request and the next acknowledge. This means if there is a request from n-1 and the acknowledge of n+1 is low, the controller N can proceed by raising its request and at the same time storing data and sending an acknowledge to the previous controller.

For the controller(n) to make a request the request from controller(n-1) must be 1 AND the acknowledge from controller(n+1) must be zero. An AND gate would be able to detect this, the output of an AND gate is only 1 when both inputs are 1. When controller(n-1) receives the acknowledge signal from controller(n), the handshake protocol dictates that it should now lower its request. This is seen by controller(n) and by the logic of an AND-gate the request signal for controller(n+1) will be lowered, and therefor the latch will be open. This is not according to protocol where it has to wait for the acknowledge signal of controller(n+1) to arrive before lowering the request. In this situation an OR gate would do the trick.

This is because the AND-gate indicates when both signals are 1 the output will be 1, but when the output is 0 no conclusions about the inputs other than at least one must be 0 can be drawn. the OR-gate is the opposite it indicates when both inputs are 0, and does not indicate more than at least one signal is one when the output is one. To solve the problem of the controller where indication of both cases is needed, a new gate is introduced. The Muller C element is a gate that is 1 when both inputs are 1, and 0 when both inputs are 0, in any other situation it holds the previous state. It is a state holding element that can be compared to a set/reset latch. The C element and its truth table are shown in 3.2. Using the Muller C element the handshake protocol will be kept in both cases. A new request will not be made before the C element has received both a request from the previous controller and an acknowledge from the succeeding controller. At the same time a request will not be released before both the previous request has been released and the succeeding acknowledge has arrived. The pipeline in 3.1 is also called a Muller pipeline. The VDHL describing the c-element can be found in appendix VHDL:Celement

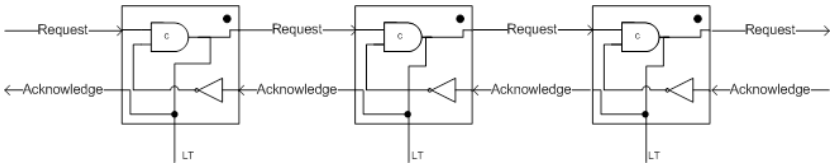
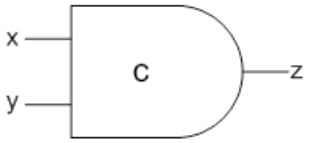


Figure 3.1: A Handshake pipeline using C element



x	y	z
0	0	0
0	1	Hold
1	0	Hold
1	1	1

Figure 3.2: the Muller C element and its truth table

3.1.2 Latch controllers

The Latch In asynchronous, design the flipflop is exchanged with a set of level sensitive latches. A flipflop which triggers on the clock edges: when it sees a rising clock edge data is copied from the input to the output and is not replaced before the next rising clock edge. A level sensitive latch is either open (transparent), data can flow directly from input to output or the latch is closed (opaque) When the latch is opaque it holds the values of the data that was on the output at the moment it closed. In this thesis the latch is in opaque state when the control signal is high or logic 1, and transparent when the control signal is low or logic 0.

Simple Latch controller A latch controller is as the word indicates a component that controls when the latch is open (transparent) or closed (opaque) The decision to close or open the latch is done by evaluating the request and acknowledge signals from the surrounding controllers. The simplest controller is the one presented earlier in the Muller pipeline. It is a circuit build from C-elements and inverters, the circuit is shown again in figure 3.4 This time including the latches controlled by the control circuit. This is the simplest implementation of a latchcontroller, but it has some obvious draw backs. Only every other latch

can hold data, this is because the input side Request and acknowledge signals are strongly coupled to the request and acknowledge signals on the output side.

To better explain the behavior it is described using a State Transition Graph (STG), STGs are a great way to capture behavior of the control circuit, and at the same time it is intuitively easy to understand. The STG matching the simple controller can be seen in figure: 3.3 the arcs represent transitions from one state to another. The dashed arcs represent signal transitions of signals from the environment, the Request and Acknowledge signals are Named R_i and A_i for the input side, and R_o and A_o for the output side. The dots marks the initial state. From the STG in fig: 3.3 and the pipeline in fig: 3.4 it becomes clear that the pipeline is full when other latch is occupied, this is because A_o must be zero, and that requires the next stage to be empty.By using a program called petrify the STG can be transformed into boolean equations the result is shown in eq: 3.1, this can also be done using State Graphs see [7] for more info on this. More advanced designs that does not have this draw back is discussed in the next sections.

$$\begin{aligned} R_o = &\quad R_i * \bar{A}_o + R_o(R_i + \bar{A}_o) \\ A_i = &\quad R_o \end{aligned} \tag{3.1}$$

Semi-decoupled Latch controller The Semi-decoupled latch controller does not have the same strong requirements from signals on the input side to signals on the output side, the controller is allowed to engage in a new handshake sequence and store new data, as soon as it sees the R_o- while A_o might still be 1. At the same time the A_i may be produced as soon as data is received independent of the state on the output side. To be able to start a new handshake sequence on the input side, while the output side has yet to complete a new internal state is added (A). see figure 3.5 This extra state is added automatically by Petrify to make sure that there is no CSC violations, (CSC - Complete State Coding, means all states have to be unique i.e may only appear once in the STG). The boolean equations from petrify can be seen in eq: 3.2 There is no equation for the A_i from the STG it can be seen that is always following A and therefore it can be omitted in the equations. An implementation using asymmetric c-gates is shown in fig: 3.6. While the Semi-decoupled latch controller has the benefit over the simple latch controller that all pipeline stages can be filled with data, it still holds a draw back. The recovery cycle on(return to zero part of the handshake) each side of the controller is still linked. From

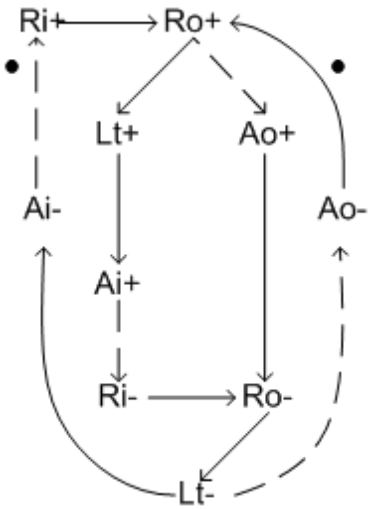


Figure 3.3: State transition graph of the simple latch controller, the dashed lines express signal events from surrounding controllers

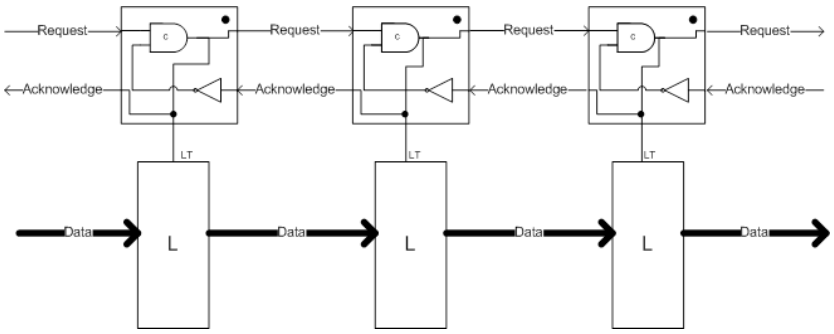


Figure 3.4: A Handshake pipeline using C element

the STG it is clear that A_i can not return to 0 before A_o has been raised.

A simulation of A fifo consisting of Simple latch controller versus one consisting of semi-decoupled controllers is shown in fig: 3.7 The simulation is made from a 6 stage deep fifo, one for each of the two types of controllers, on the input is an eager stage that feeds the next Req signal as soon as the first handshake sequence is complete. On the output end of the pipeline is a lazy consumer,

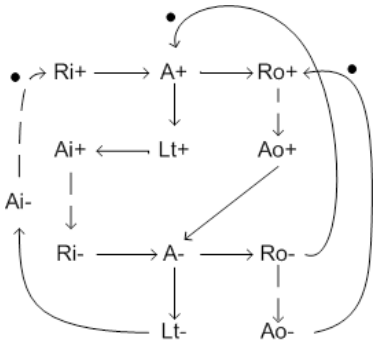


Figure 3.5: State transition graph of the semi-decoupled latch controller, the dashed lines express signal events from surrounding controllers

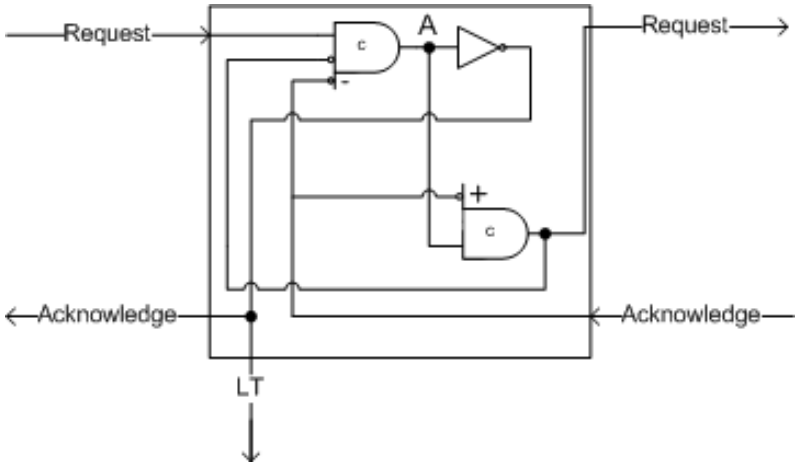


Figure 3.6: Semi-decoupled Latch control Using asymmetric c-gates

it does not response to re request signal of the last fifo stage, which means at some point the fifo will be full and stall. From Fig:3.7 its clear that the Fifo made from simple latch controls stall after it has consumed 3 request signales (tokens) leaving every other latch not holding data. On the other hand the Fifo made from semi-decoupled controllers continues on until every latch holds data.

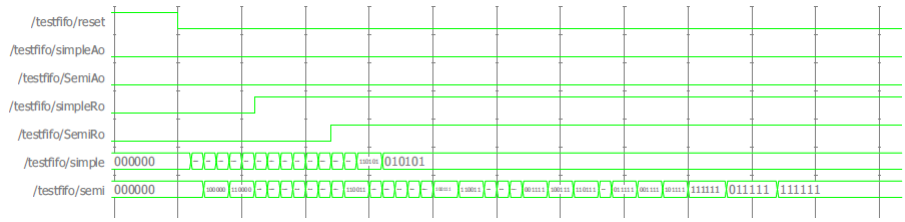


Figure 3.7: Comparison of 6 stages deep fifos one made from simple latch controller, and one made from the semi-decoupled controller. The outputs are the simple/semi signals. where the output represents the state of each latch from controller nr 012..5, a one means the latch is holding data where as a 0 means that the latch is not holding any valid data

$$\begin{aligned} A+ &= R_i * \bar{R}_o \\ A- &= \bar{R}_i * R_o * A_o \\ R_o+ &= A * \bar{A}_o \\ R_o- &= \bar{A} \end{aligned} \tag{3.2}$$

Fully-decoupled latch controller The fully-decoupled latch controller further relaxes the coupling from input side to output side, and removes the coupling between the recovery cycles. The Decoupling is accomplished by inserting another internal variable B. The resulting controller has input side handshake and output side handshake that can run concurrent, which results in a very complex STG se fig: 3.8 the Resulting boolean equations from petrify is shown in eq: 3.3 again the latch control signal is always A therefore it has been removed from the equations. Also from the equations it is obvious that the only thing changed compared to the semi-decoupled controller is the equation for A_i in the semi-decoupled case this was always following Lt, and therefore following A, in the fully-decoupled controller it depends only on the R_i and internal variables.

The choice of controller used in this thesis is the semi-decoupled one, this is from the conclusion that the fully decoupled one is far more complex to implement and will result in an increase in area overhead that does not justify the potential performance gain. In [7] It is shown that the fully decoupled is almost twice as big in area, where the processing pipe time for a semi-decoupled controller is 54.7ns and for the fully-decoupled controller it is 37.7ns only a gain of approximately 30%, also the article notes that in FIFO applications the gain in performance is not noticeable from the semi-decoupled to the fully-decoupled

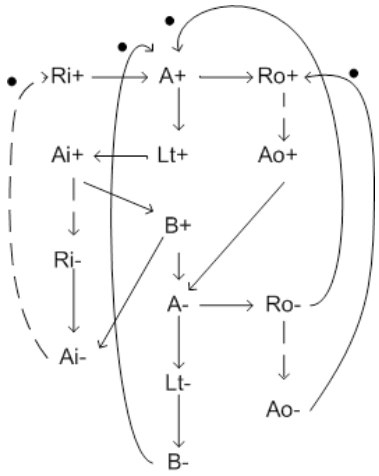


Figure 3.8: STG of the fully decoupled latch controller.

controller.

$$\begin{aligned} A+ &= \bar{B} * \bar{R_o} * R_i \\ A- &= B * R_o * A_o \\ B+ &= A_i \\ B- &= \bar{A} * \bar{A_i} \\ R_o+ &= A * \bar{A_o} \\ R_o- &= \bar{A} \\ A_i+ &= A * \bar{B} \\ A_i- &= \bar{R_i} * B \end{aligned} \tag{3.3}$$

Master / Slave Design Each flipflop is substituted with a Master and a Slave latch, which closely resembles the behavior of a flipflop (flipflop is contructed from two levelsensitive latches) some synthesis libraries have flipflip where control signals for both latches inside is available, this design can be used with an advantage since this design is the most compact area wise. Another important reason for using a double latch design is that with the master/slave design at least one is opaque in any situation. If only one latch is used for each flipflop a situation where all is open can take place, when a combinatorial block

is calculating the next value the output of this block can change several times before ending at the correct output, with all latches open this would result in lots of very long wires being charged and discharged for no reason, and this can be very expensive in terms of power. see fig: 2.3. The master and slave latch needs to be initialized in opposite states so that one is holding and the other is transparent for the design to function correctly.

Latch controller Behavioral and synthesizeable model To be able to simulate and test the desynchronized designs, a behavioral of the latch controller have been implemented in VHDL from the boolean equations in eq: 3.2 The component is very simple and only mimics the behavioral of a latch controller without any timing assumptions. The VHDL for the behavioral can be found in Appendix C.3.3 after the correct behavior of a desynchronized circuit is verified, a synthesizable model is needed for implementation. In article [2] a design using only basic blocks is presented. At the same time the design presented incorporates the master/slave design we want. The implementation is shown in fig: 3.9 its the same as the one in the article with the small difference that the inverting on the latch control wires is removed, this thesis used latches that a opaque with control level 1, the article uses latches that are opaque with latch control level 0. The implementation is done and tested in section:3.2.

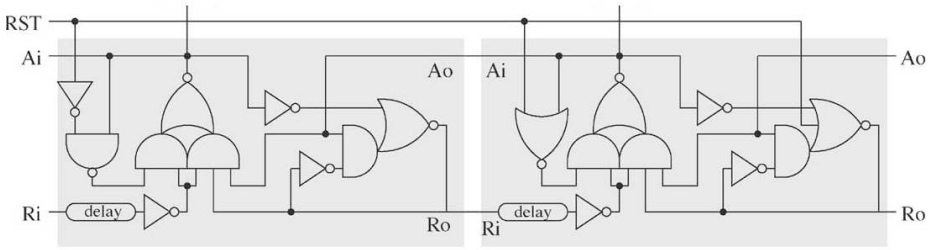


Figure 3.9: Synthesizeable model of the semidecoupled latch from [2]

3.1.3 The matched delay

Completion Detection In synchronous design the clock serves as completion detection, it is expected that all combinatorial stages has finished before the next clock period. The clock period is fixed, and all stages has the exact same amount of time to finish.This time is based on an timing analysis of the slowest

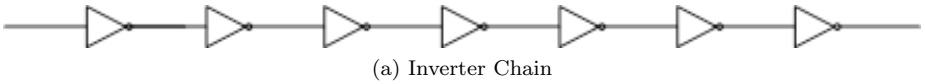
stage in the design. There is no clock indicating when a stage is finished in asynchronous design, this is done by the handshaking signals. The data is often affected by some combinatorial delay caused by the combinatorial logic through which it must pass, the handshake signals does not pass through the same logic. Therefore it is vital to be able to predict the delay through a given stage, so a matching delay can be inserted to slow the handshake signal indicating the completion of a given stage. When delay matching a delay in inserted in to the handshake protocol that matches the delay of the combinatorial circuit between the latches controlled by the two controllers. It is crucial that the data is valid before a latch closes therefore the minimum delay of the element inserted should be equal or greater than the worst case delay of the combinatorial path.
If Handshake signals are routed on the chip as a bundle with the data, they will experience the same variations an will therefore track the delay through the combinatorial path very precisely. In the Push channels used in this thesis the delay element is always placed on the Request wire.

A simple delay element one method to implement a delay element is an inverter chain with n number of inverters to reach the desired delay. shown in fig: 3.10a The drawback of this simple delay element is that the delay affects both the data transfer and the return to zero period. But only the delay of the rising request signal indicating that data is valid is necessary, since the return to zero period does not indicate any data transfer in the combinatorial path. Alternatives to the simple implementation is shown in fig: 3.10b and 3.10c the advantage of these two is only the rising edge of the signal is affected by the delay, The chain of AND-gates can if a large delay is needed have a very large fanout which could be a problem, the mixed-gate chain from[2] has half the fanout and is implemented with standard inverting C-mos logic, and is therefore preferred for this thesis.

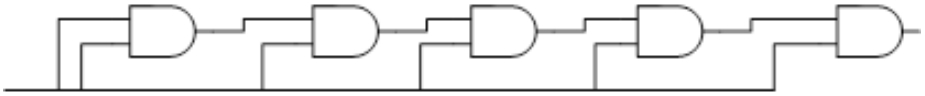
A short test of an implementation of mixed-gate implementation is shown in fig: 3.12 the return to zero delay is obviously a lot less than the rise delay.

3.1.4 Forks, Joins, Multiplexers and De-Multiplexers

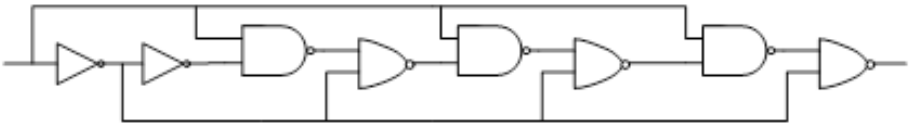
Forks and Joins Two very important components when dezynchronizing is forks and joins, these are used to keep track of $1 : many$ and $many : 1$ handshaking. i.e to synchronize multiple datapaths, The fork synchronizes multiple outputs and the join does the same with inputs. This is f.ex important when feeding inputs to an adder the addition must not start until both inputs have arrived. In order to synchronize the Muller c-element is used, in a fork the



(a) Inverter Chain



(b) Chain of And gates



(c) Chain of mixed gates

Figure 3.10: three ways of implementing a delay element

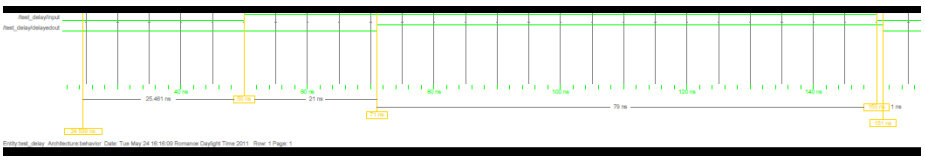


Figure 3.11: Simulation of an unbalanced delay element, the rising edge is delayed approximately 20ns while the falling edge is only delayed by 1 ns

request signal is simply split and sent to the n-controls that needs it and c-elements are used to synhcronize the acknowledge signals confirmation that all receivers have stored the data. The join is the opposite, the c-element is used to wait until all input controllers have data ready, then the request is asserted for the next stage, when this have stored the data the acknowledge signal is simply split into n signals for the input controllers. Only a synthesizeable model of the fork and join have been implemented, since this is straight forward using the synthesizeable C-element. The VHDL for these two components can be found in appendix C.1.2

Multiplexer and De-multiplexer In some situations it is necessary to guide a signal to one of multiple receivers, while the others are left idle. A compo-

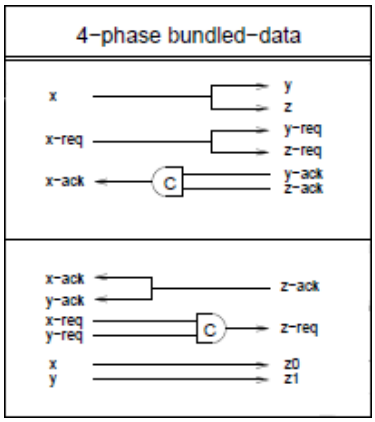
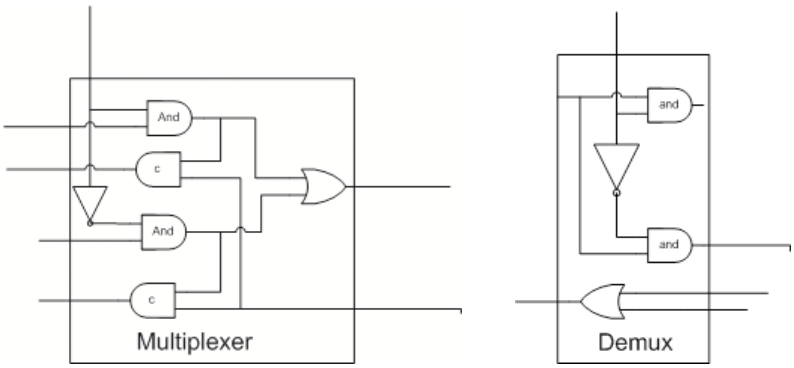


Figure 3.12: Implementation of the Fork and Join using a C-element figure taken from [6]

nent that has this functionality is the the De-multiplexer. It forwards the input request and the data to an output selected from a control signal. The Multiplexer is the opposite it Select one of multiple inputs and forwards it to an output decided by a control signal. The other inputs are ignored and might have requests pending, these will be stalled until the control signal decides to forward this request. This functionality is an excellent way of disabling some latch controllers keeping the respective latch at its current output by guiding the incoming request via another path.
Asynchronous implementations of the two can be found in [**?**] but these cannot be implemented directly in the examples used in this thesis. In desynchronization the controllers for the multiplexers and de-multiplexers are already implemented as combinatorial logic to use these control signals the components must be modified slightly. The de-multiplexer shown in fig: 3.13b does not use any c-elements this is because the request out should return to zero as soon at the control signal changes or the input request i reset. This insures the transparency to the handshake controls.
In the multiplexer 3.13a the input is chosen by an AND of the control signal and the two request signals. The acknowledge is generated by a c-element of the internal request and the acknowledge signal. The c-element ensures that the acknowledge in is held high until the acknowledge out is reset to zero.



(a) Modified multiplexer (b) Modified de-multiplexer

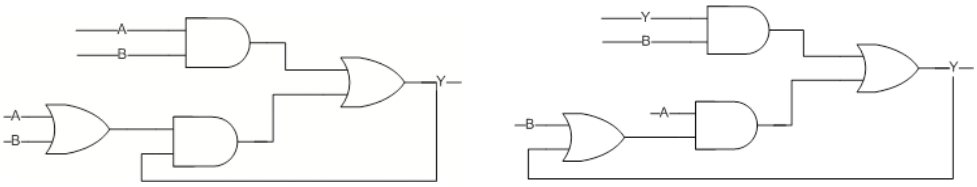
3.2 Synthesis

Synthesis of the C element The muller c element is a special component and is not included in standard synthesis libraries, so before the element can be implemented, the synthesized model must first be verified. Using synopsis to compile and synthesize the component. After synthesis the behavior is the same, but to be certain the Verilog netlist of the synthesized module is checked.

```
1 module Celement( A, B, Y)
2     input A,B;
3     output Y;
4
5     A05NSVTX1 u1 (.A(A), .B(Y), .C(B), Z(Y));
6 end module;
```

A short run through of the above netlist: The module Celement has the inputs A & B and the output Y, the component i instantiated using a library component called A05NSVTX1 the SVT in the component name is indicating a standard cell is used. The standard cell is a average model between the hight threshold and low threshold cells. The A05NSVTX1 component has three inputs and one output (this can be seen in the library file) instead of listing the Verilog component file, the schematic of the Verilog instantiation is shown in fig: 3.13b the schematic of the VHDL is shown in fig: 3.13a

The two gate implementations are very similar but not identical, so the behavior is verified from the truth tables below, clearly the behavior is identical. So the synthesis of the c element is complete and there should be no problem implementing the C element in designs.



(a) gate model of the VHDL C element (b) gate model of the Verilog C element

Figure 3.13: The C element before and after synthesis, the implementations are similar but not identical.

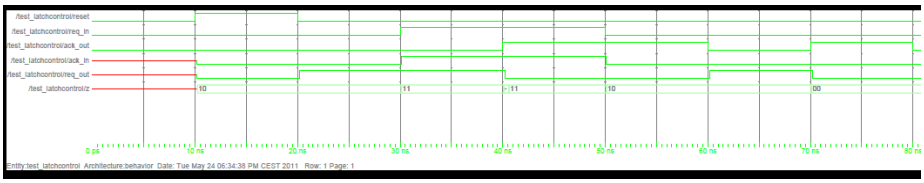


Figure 3.14: Simulation of the Latch controller after synthesis.

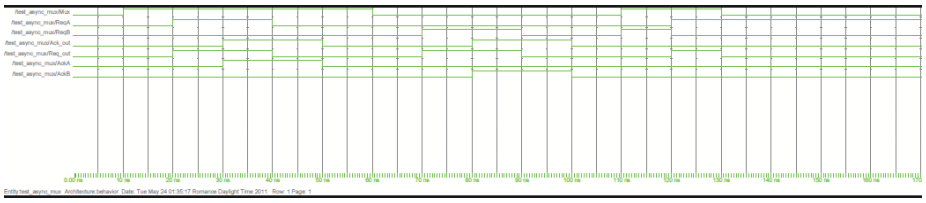
Celement					Verilog			
A	B	yin	y		A	B	yin	y
0	0	0	0		0	0	0	0
0	0	1	0		0	0	1	0
0	1	0	0		0	1	0	0
0	1	1	1		0	1	1	1
1	0	0	0		1	0	0	0
1	0	1	1		1	0	1	1
1	1	0	1		1	1	0	1
1	1	1	1		1	1	1	1

Synthesis of the double latchcontroller After synthesis the behavior of the latch control is verified using modelsim, and the synthesized netlist. It is very important that signal constraints are intact after synthesis. This is done in the same way as with the c element, the behavior is shown from simulation post synthesis in fig: **??** . The gate implmentation is the same as in the one presented in 3.9.

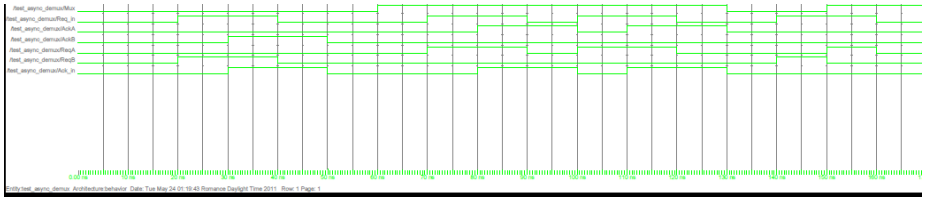
Delay element The Delay elements cannot be synthesized using synopsis, the tool will trim the delay chain since the input and output are the same, and synopsys treats this as overhead in area since there is no logic function. A work around to this is to make the netlist of the delay by hand, and insert it

into the desired points in the synthesized netlist. This is not straight forward but can be done. The easiest way is to define a component with no logic just an input connect to an output, synopsis will keep the structure of the design. After synthesis locate the dummy box and insert the Verilog code for the desired delay. These files are very long (a chain of 20 gates gives approximately a delay of 2ns) Therefore the Verilog file is not presented here.

Multiplexer and De-multiplexer Both of these components have some timing requirements on the control signal. the control signal must be stabile throughout the complete handshake sequence, it cannot change until the handshake signals on the input side have returned to zero. The problem is illustrated in the two simulations in 3.15.



(a) Simulation of mux, the problem of the control signal can be seen at 130ns



(b) Simulation of demux, the problem of the control signal can be seen at 150ns

Figure 3.15: modified asynchronous multiplexer and de-multiplexer

3.3 Design flow - The steps of desynchronization

This section will describe the design flow of desynchronization, From Synchronous RTL level description in VHDL until synthesized design. The Process takes several steps from recoding the VHDL, locating registers, substitution with controllers, inserting forks and joins etc. Till synthesize and floorplanning, and finally simulation and verification of the design. Also a simple comparison between the synchronize circuit and the desynchronized equivalent is done.

3.3.1 Optimizing VHDL for desynchronization

Compilers today are optimized for synchronous design, this makes it very easy to describe synchronous design i VHDL using simple expressions like

```
1 if(clock'event and clock='1')\\
```

the compiler immediately recognize this as a flipflop. The latch equivalent would be:

```
1    if(control='0')\\
```

omitting the else clause will result in an inferred latch, which is good enough. A good way to start desynchronization of a design is to take the register transfer level(RTL) schematic and locate all registers. In most VHDL designs today the registers is filtered into the combinatorial logic as shown here:

```
1  count : process(clk,reset,pause)
2        begin
3                        if clk = '1' and clk'event then
4                                if reset ='1' then
5                                        tempcountLow <= "00";
6
7                                        tempcountHigh<= 0 ;--tempcountHigh;
8                                elsif pause ='1' then
9                                        tempcountLow <=tempcountLow ;
10                                       tempcountHigh<= tempcountHigh;
11                               elsif tempcountLow ="10" then
12                                       tempcountLow <= "00";
13                                       if (tempcountHigh = 89 ) then --90
                                              column= 0-89
14                                                tempcountHigh <= 0;
15                                       else
16                                       tempcountHigh<=tempcountHigh+1;
17                                       end if;
18                               else
19                                       tempcountLow <= tempcountLow+1;
20                                               tempcountHigh<=tempcountHigh;
21                               end if;
22                       end if;
23
24       end process;
```

This is a real counter used en a later chapter 6, its intuitively easy to understand for a VHDL designer, the only problem is its not very easy to desynchronize. The counter is obviously two registers one holding the lowcount value and one holding the high count value, both is fed to some combinatorial circuitry that calculates the next count values. the RTL schematic is shown in fig: 3.16a

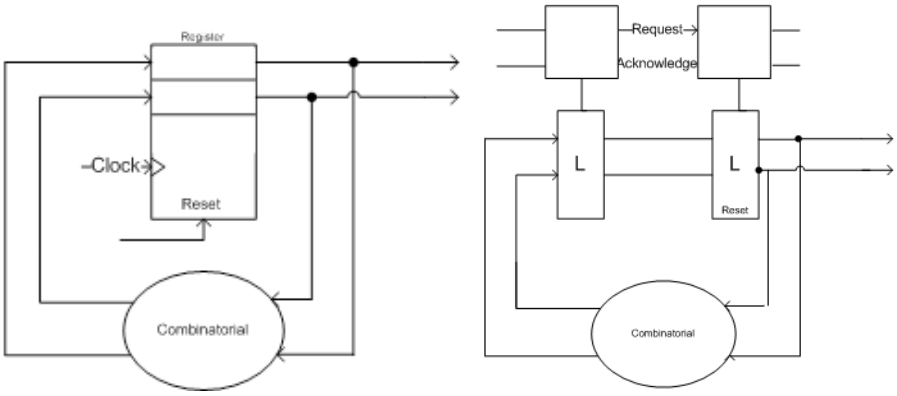
looking at the Schematic, the substitution of registers with latches seems straight forward, done in fig: 3.16b, The VHDL code is not so straight forward. Instead of implementing somesort of modified latch in every component its easier to recode the VHDL into a two process structure. The sequential logic is put into one process and the purely combinatorial is put into another. Demonstrated in the VHDL code here :

```
1  count :  process(clk,reset)
2                                  if reset ='1' then
3                                          tempcountLow <= "00";
4                                          tempcountHigh<= 0 ;--tempcountHigh;
5                 elsif clk = '1' and clk'event then
6                       tempcountlow <= nextcountlow;
7                       tempcounthigh <= nextcounthigh;
8  end process;
9  combi : process(tempcountlow, tempcounthigh,pause)
10         begin
11
12                                 elsif pause ='1' then
13                                         nextcountLow <=tempcountLow ;
14                                         nextcountHigh<= tempcountHigh;
15                                 elsif tempcountLow ="10" then
16                                         nextcountLow <= "00";
17                                         if (tempcountHigh = 89 ) then --90
                                                column= 0-89
18                                                 nextcountHigh <= 0;
19                                         else
20                                         nextcountHigh<=tempcountHigh+1;
21                                         end if;
22                                 else
23                                         nextcountLow <= tempcountLow+1;
24                                     nextcountHigh<= tempcountHigh;
25                                 end if;
26                         end if;
27
28         end process;
```

Now the Register is in its own process and can be substituted by latch code or by a latch component which we shall se in a later chapter. An alternative to the two process structure is to create the combinatorial circuit as one component and the register as another. This takes more time but makes desyncronization even easier.

3.3.2 Substitution of Registers - The double latch design

The double latch design For easy implementation, and to avoid doing the same routing over and over again the master and slave latches are combined into one component this saves time since now only the to control signals have to be routed, the in and outputs of the component can be connected directly to the wires previously connect to the register. The latch is generic so that it can be used for all purposes.



(a) synchronous counter (b) asynchronous counter

Figure 3.16: RTL of synchronous and asynchronous counter (asynchronous not complete)

Double latch controller As described the latches are combined two and two in a master/slave design, where the slave is initialized opposite the master latch. The controller design chosen for the implementation is already a double controller design, that controls both the master and the slave and also handles the initialisation. Therefore it makes sense to make a behaviorial for testing also as double latch controller. The behavioral is just a component that combines two identical copies of the latch controller behavioral implemented with the boolean equations from eq:3.2.

Register substitution Now each register can be substituted with a double latch component and a double latch control component, the latch inputs and output are simply connected to the datapath, and the two control signals are connected to the controller.

Inserting forks and joins Following the Data path from one latch to the next from input to output, every time data is split into multiple latches a synchronizing fork is inserted in the handshake signal at the same place, and every time a latch receives data from multiple latches a synchronizing join is inserted.

Inserting matched delay elements Between each latch controller a delay is inserted, for functional testing a simple after statement will do, it is recom-

mended to implement the after as generic delay block, for easy substitution with the actual delay block, at this point it is not important to calculate actual delay values, as long as the delay is long enough.

Verifying the design The desynchronized circuit behavior is checked by simulation the design using modelsim and the original test bench, when the functional behavior has been confirmed to be correct, all delay elements should be substituted by a simple wire (the after statement does not synthesize)

If the design delivers data to environment a consumer needs to be added before the design can be tested. this is done very simple by feeding the request out of the circuit to the Acknowledge of the same channel after a small delay. also if the design takes inputs from the environment a provider needs to be added at the input, this can be done by feed the inverse Ack in to the request in after some delay. this prevents the design from stalling due to lack of empty latches.

Synthesizing The design This thesis uses synopsys design compiler for synthesis, its recommended to use design vision to understand how the synthesis is done the first couple of times, design vision is a graphical interface to desing compiler. A script for easy synthesis have been created this is easier to use when comfortable with the synthesize process, and can be found in B.1
From synthesis an SSTA evaluation of the delays for each block can be found. Using this file the delays can be easily inserted. Resynthesize the files including the inserted delays, its crucial under synthesis that some of the components are not altered, and synopsis will trim the delay element away if not constrained, this can be done by using the dont touch feature. After synthesis the floorplan can be run also using design vision or a script found in refsyn:floorplan The synthesis is now completed, to get info on how to print all the needed reports, and info about the constraint settings like don't touch and timing check out the guide in A

Simulation of the synthesized design After synthesis the design can be simulation in a mixed simulation using the verilog netlist from synthesis and the VHDL test bench from the original circuit.

Comparison Finally a comparison on the switching activity can be done using the developed matlab file found in B.3

In this chapter all the basic components needed for desynchronization was presented, and explained in detail. The behavior after synthesis was verified. A step by step guide to desynchronization was presented. The potential problems of the multiplexer and de-multiplexer was identified. Also how to modify the VHDL code for easy desynchronization was shown.

Chapter 4

Example 1 - Accumulator (Accu)

In this chapter and the next two the theory presented in the previous chapters will be put to practice, also the steps of desynchronization presented in 3.3.2 will be demonstrated. The first example is a simple test circuit constructed for this thesis, an Accumulator that simply takes the input and adds it to the sum of the previous inputs. The second example is still a simple circuit but it is an real design, calculation the greatest common devisor of two inputs, the last example is significantly bigger in size and complexity, this is an Edge detection algorithm, designed to work as a hardware accelerator on a bus shared with a general purpose processor. In all examples the 4-phase handshake protocol is used, and all channels are push channels. This means the delay insertion will always be on the request signal.

4.1 Synchronous Accu

To test the desynchronization theory we start with a very simple design. An 8 bit Accumulator which simply takes an input and add it to previous inputs, have been designed for the purpose. A Schematic of the synchronous design can

be seen in 4.5 and the VHDL code can be found ind Appendix C.2
The behavior is shown in fig 4.4. The value 1 is the first input, the corresponding output will therefore be one($0 + 1 = 1$), the next input is 3, the corresponding output is 4 ($1 + 3 = 4$) and so forth. the formula for the output of the Accumulator where x_t is the input x at time t and y_t is the output y at time t is:

$$y_{(t+1)} = x_t + y_t \tag{4.1}$$

A simulation of the behavior can be seen in fig: 4.1 showing the behavior is the same as in 4.4

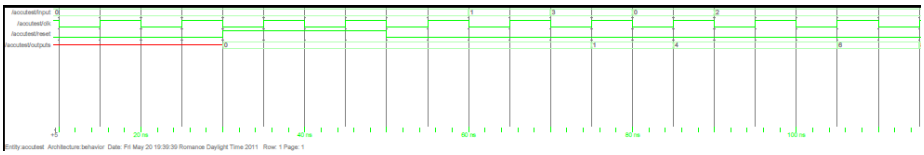


Figure 4.1: Behavioral of the synchronous accumulator

4.1.1 Desynchronization of Accumulator

The first attempt of desynchronizing the accumulator is following the desynchronization steps from Chapter 2 to create a asynchronous behavioral of the synchronous accu, and verify that the functional behavior is the same.
The first step is to recode the VHDL in such a way that the registers are in their own processes, this is not necessary since the VHDL is constructed with the purpose of desynchronization, the register is already separated from the combinatorial network.

4.1.2 Steps 2 - 6

Register substitution Replacing all registers with a double latch component and a controller is straight forward using the components designed for the purpose, these are described in detail in chapter 3.
The challenge is to place the join and forks where needed, to find the places where either a join or a fork is needed follow the data from input through the registers and the adder to the output. The first thing to handle is the communication with the environment starting from the input there the environment needs to indicate when new data is available and should be processed, and the input register should be able to send an acknowledge back telling that the data

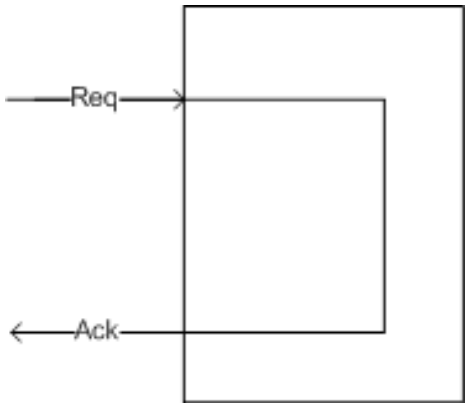


Figure 4.2: Simple way to implement the eager consumer, the Request out is returned as an acknowledge.

has been received. So the clock in topfile of the accumulator is replaced by a request$_i$ and an acknowledge$_i$, The component also has an output with the result of the accumulation, since this is purely testing there is nothing receiving the output data so there is no need to implement a handshake channel on the output side, instead an eager consumer is added, this will consume all data and handshake signals instantaneously and return the required acknowledge. With the very eager consumer the accumulator will work at its top speed. A very simple implementation is shown in fig:4.2.

Joins and Forks From the input register the data flows through the combinatorial adder which takes two inputs, the input x, and the previous output y, for the accumulator to calculate the correct result it is vital that the inputs are both stabile before the result is saved in the output register. So clearly a join is needed here to synchronize the two inputs into the adder. The output of the adder is fed directly to the register no forks or joins needed here, but on the other side of the output register, the output serves both as an output of the component and as an input of the adder, to split the request signal and synchronize the acknowledge signals from both receivers a fork is needed.

Inserting temporary delay elements The last thing before the a desynchronous behavioral is ready for test is the insertion of matched delay elements. Again following the datapath in the Synchronous design, there is only one path where the data passes some combinatorial logic this is in the adder, the delay

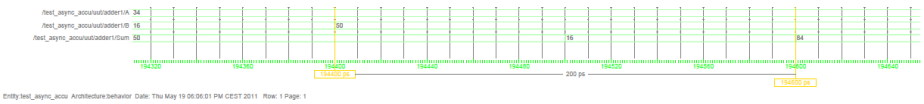


Figure 4.3: The Delay from last input arrives at the Adder till the result is present at the output is only 200ps

is not known in advance so a delay that is "big enough" is inserted, a delay of 8ns should be suffice. The delay should mimic the delay from input to output of the adder so the insertion point must be from data is ready on both input of the adder, hence between the join and the output register.

4.1.3 The desynchronized Accu

After desynchronization the schematic look like 4.6, the functional behavior is as expected the exact same as the synchronous, the simulation is shown in fig:4.7 in the simulation is also the waveforms of the latch controls to show how the handshake propagate through the circuit. The next step is to synthesize the design using synopsis design compiler, the first synthesis is to get the actual delay through the combinatorial path. This delay can be read from the timing report of the synthesis and is shown in a simulation of the synthesized adder in fig:4.3 From the last input arrives at the adder until it produces the correct result is only 200ps, so a very short delay indeed. The Matched delay is adjusted to 300ps, to give a little slack and the circuit is synthesized again.

Simulating the synthesized design After the matched delay have been inserted, and a new synthesis have been completed, the resultant Verilog netlist is synthesized using mixed-mode simulation in modelsim, the test bench used to test the behavioral together with the synthesized netlist and the library files of the used standard is needed. The result of the simulation i shown in fig: 4.7. The simulation clearly shows the same behavior as the synchronous circuit, the timing is a little different, since the speed of the asynchronous is determined by the delay in the latch controllers and the inserted delay element and also by how fast new input is presented, And the speed of the synchronous is determined from the clock period, which has not been set for maximum speed.

Figure 4.4: Behavior of accumulator

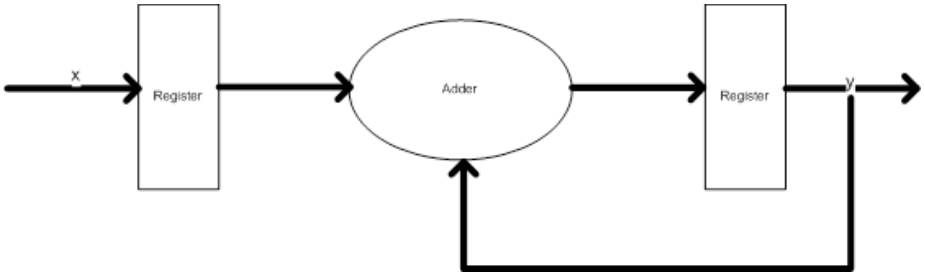


Figure 4.5: Schematic synchronous Accu

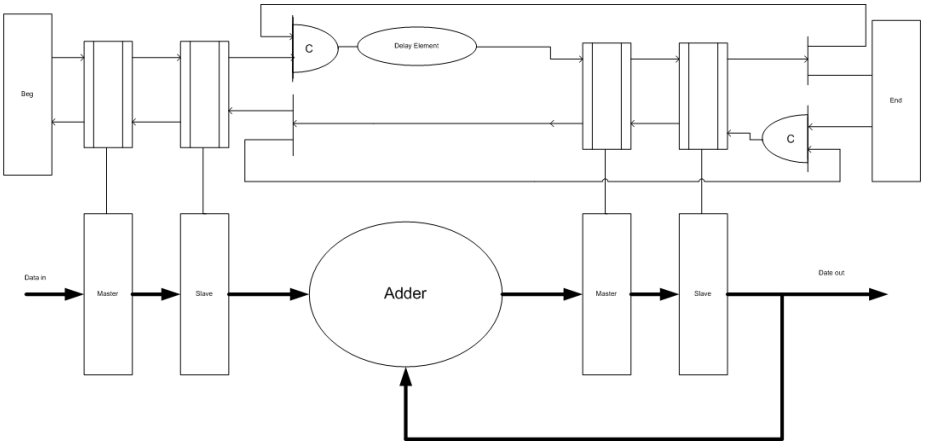


Figure 4.6: block diagram asynchronous Accu

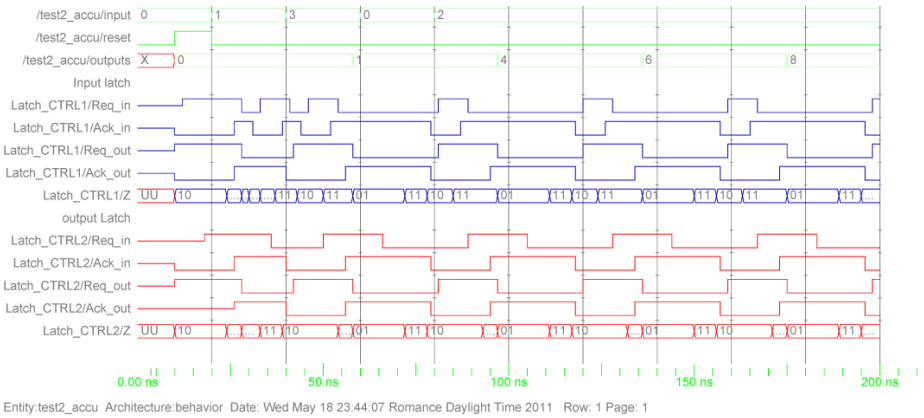


Figure 4.7: Simulation of the behavior of the desynchronized accumulator

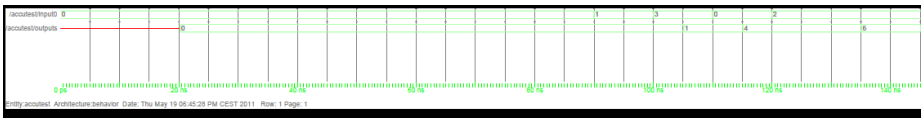
4.2 Test and results

4.2.1 Post synthesis measurements

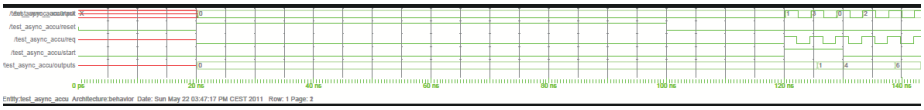
After simulation of the synthesized design it is time to compare the differences in the two different implementations of the same circuit. To do that a value change dump file (VCD file) have been created using modelsim for both the synchronized and the desynchronized design. Besides the VCD file the report from the synthesis containing nodal-capacitances is needed. The comparison of these two is not to compare the latency or throughput of the circuit, but to evaluate the switching activiy, and hence the EMI noise and current peaks. To be able to compare these features a Matlab file have been developed and can be found in appendix B.3 the file takes the VCD file as input and the capacitances for each node. It then locates the switching of each node and multiplies it with the node capacitance. this gives a good estimate of the current consumption and EMI noise at the given time t.

To be able to analyze the results from the matlab script, the two simulations of the synthesized circuits, they are shown in

In fig: 4.8 It can be seen that the actual calculations starts late in the simulation, approximately after 100ns. The idea is this should be obvious in the asynchronous measurements that there is no switching activity until the calculation start, where as the clock will be ticking in the synchronous version.

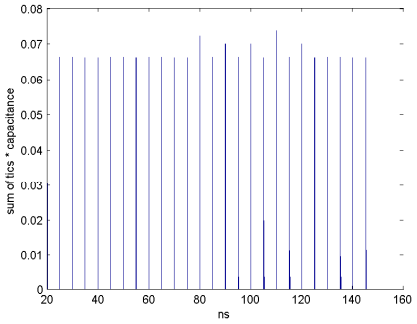


(a) Synthesized synchronous accumulator behavioral

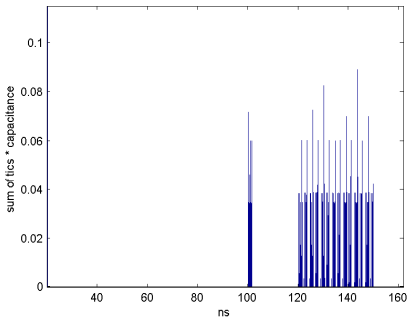


(b) Synthesized asynchronous accumulator behavioral

Figure 4.8: The simulation after synthesis, of the two implementations



(a) Matlab result of the synchronous accu



(b) Matlab result of the asynchronous accu

Figure 4.9: Matlab result of the two simulations

Both simulations are processed with the matlab script created for the thesis the result is shown in fig:4.9

In the fig: 4.9a it is obvious that the clock ticks are very expensive, and that even without any calculations the clock keeps ticking. In the asynchronous version there is no switching activity until the reset is released after 100ns, The spikes in both figures are when the reset is released and when the output of the adder changes do to the inputs. From the analysis of the two simulations it seems that the asynchronous has higher switching activity, but the overall average of the spikes are lower than the average in the synchronous ones this mean a reduction in the current spikes and EMI noise. At the same time the circuit is idle until inputs are presented, which means it does not consume any power. This is a very simple circuit, and the latch controls and the fork and joins results in more than 50% area increase which can explain the massive increase in switching activity.

Area Reports			
	Sync	Async	increase
Combinatorial	144	263	82%
Non Combinatorial	298	421	41%
Total Area	443	685	55%

Chapter 5

Example 2 - Greatest common divisor(GCD)

This example is about finding the greatest common devisor between two numbers. The greatest divisor means the highest positive integer than both numbers can be divided by without leaving a remainder. An example the greatest common divisor between 156 and 30 is 6. This is usually written $GCD(156, 30) = 6$ the rest of this chapter will first be presenting the behavior of the synchronous GCD, second the first attack at desynchronizing the design is presented the first attempt is to get a basic desynchronized version, the second attempt will look into the question of granularity, and try to answer the question of how fine grained the desynchronization should be.

5.0.2 The synchronous greatest common devisor

There are several methods of calculating the GCD of two numbers, the implemented algorithm uses a very simple iterative method where the largest of the two numbers is replaced by the difference of the two, this is repeated until the two numbers are equal, when this happens the greatest common divisor have

been found. The algorithm is shown in 5.1

$A = 56$	$B = 12$	*Calculation*
A	B	$56 - 12 = 44$
44	12	$44 - 12 = 32$
32	12	$32 - 12 = 20$
20	12	$20 - 12 = 8$
8	12	$12 - 8 = 4$
8	4	$8 - 4 = 4$
4	4	$4 = 4$
	$GCD(56, 12) = 4$	

(5.1)

The synchronous GCD design presented is taken from an assignment in the course 02154 at DTU. and the RTL can be seen in fig: 5.1
The VHDL implementation is done in a two process way with one process for the three registers(register A, B and nextstate) and a process for the combinatorial circuit(evaluate and subtract) The synthesized synchronous GCD behavior can be seen in and the VHDL code can be found in C.4.

5.0.3 Desynchronization of GCD

In the GCD the registers are taken out into their own component for easy substitution, The internal states is defined as type state in the VHDL:

```
1 type state_type is (WAIT_A,SET_ACKA,WAIT_B,RESET_ACK,EQUAL_CHECK,
      A_GREATER_CHECK,WRITE_A);
```

This poses a problem when the state signal is needed outside the component, therefore this signal must be recoded into a binary version. The recoding of the states can be done with 3 bits, (there are 7 states)

WAIT A	000
SET ACKA	001
WAIT B	010
RESET ACK	011
EQUAL CHECK	100
A GREATER CHECK	101
WRITE A	110

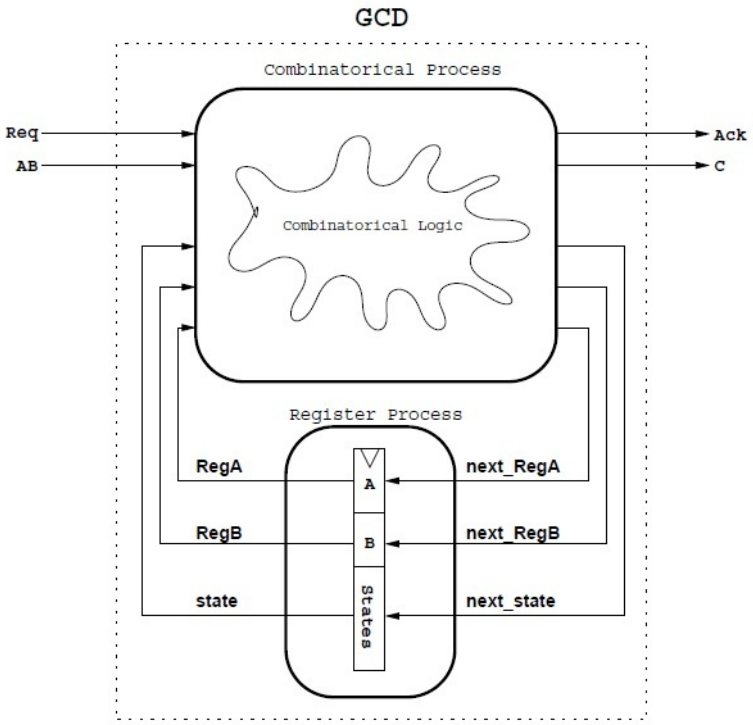


Figure 5.1: RTL of the synchronous GCD

5.0.4 Steps 2 - 6

Since the registers are now i their own component it is simple to substitute the registers with latches and a control component. In fig: 5.1 there is already a request and an acknowledge signal, these are used to indicate when new data is ready at the input of the GCD and when the result have been computed at the output, they are not handshake signals in the asynchronous sense, the request at the input does not return an acknowledge after an input have been received, and the acknowledge at the output is not an indication that the component receiving the result has actually received anything. The request and acknowledge signals in the synchronous GCD should be thought of as a start and a finish signal and nothing else.
To be able to communicate with the surroundings the clock signal is substituted with a handshake channel (set of request and and acknowledge signals) at the input and at the output.

Forks and Joins At this first approach there is only one register that holds both A, B and Next state, this makes it fairly simple. From the synchronous schematic the combinatorial logic takes two inputs, the input from the environment and the previous calculated results, as in the accumulator example this indicates the need of a join to synchronize the inputs. The output of the combinatorial is connect both to the input of the logic and to the output of the GCD component this as we know require a fork.

The Delay element From the synchronous simulation it can be seen that the calculation of each stage takes about 5ns, to be on the safe side a delay of 10 ns is inserted. The first behavioral version of a desynchronized GCD is now complete and is shown in fig: 5.2

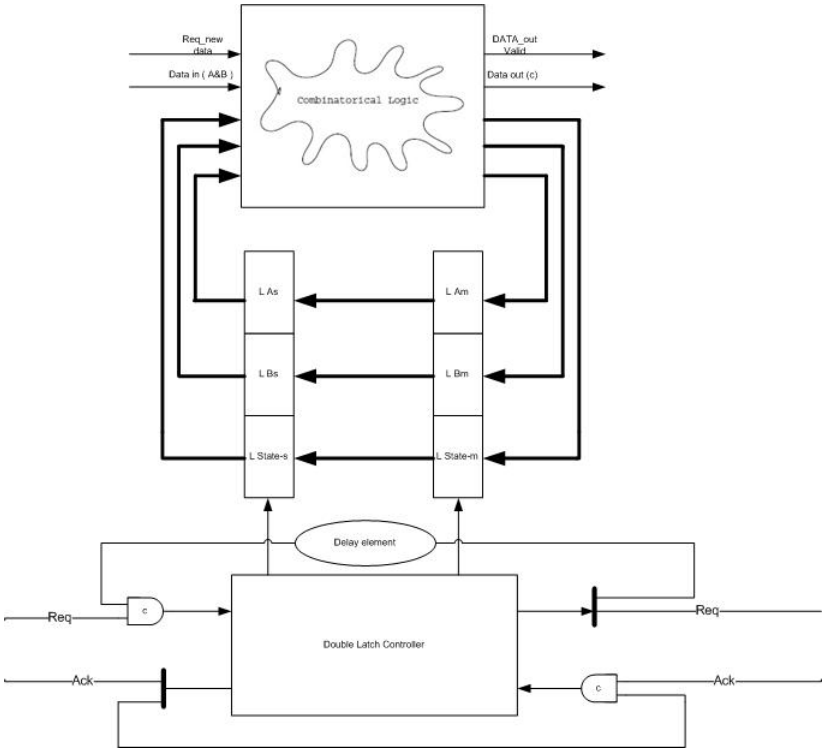


Figure 5.2: RTL of the asynchronous GCD

Simulation of the first behavioral

In the simulation 5.3 the Behavior is correct although running a lot slower than the synchronous version. Since this is just a behavioral the speed is not of interest at this moment. What is on the other hand interesting is two observations. The GCD handshakes with the environment at every calculation step, but no data can be given to the GCD until it has finished calculating, and the data coming from the GCD is not valid! Every time the GCD handshakes with the environment with out data being handed over, the circuit uses unnecessary energy, we will try to solve this problem soon. Another observation is the first version of the desynchronized GCD uses one big register to hold three kinds of data. It might be possible that not all data changes at every handshake. From eq 5.1 only one of the variables A or B is updated at any given calculation Again updating an unchanged register is a waste of energy, but by implementing three separate controllers, a lot of overhead is introduced and the power used by these might be greater than the power saved. This problem will be investigated at the end of this chapter.

Handshaking when needed First the problem addressed is at the output channel, there is no need to pass invalid data on to the output of the GCD, this problem can be solved by inserting a decision component, where the Request out is simply passed back as an acknowledge until the finish signal indicates that the GCD calculation has finished, then the request and acknowledge signals are connected to the output. This action can be implemented with gates as shown in fig: 5.4b

The input is slightly more difficult, the GCD should accept two input values, and then stall handshaking until calculation is done, the protocol for the GCD such that there will be an ack out on the calculation done signal when the first value is received. This can be used as an ok signal to load another value, just as it is used to send the request out from the GCD. The problem is the request signal in to GCD will stay at logic 1 until an acknowledge has been replied by the GCD, this will stall the join. The way to solve this i pretty straight forward. Just as in the case at the output the request in signal is combined with the inverse of the acknowledge signal via an AND gate. this will allow the request in to stay at 1 waiting for the acknowledge out. The acknowledge out should be stalled until the calculation done is raised, this is done by a simple AND gate. The input logic design can be seen in fig: 5.4a
the behavioral have been tested using these two components, but since there is no environment they are removed before creating a synthesizeable version.

5.1 Fine grained design - splitting the registers

The observation that only one register is updated at any cycle raises the question, if something could be gained from splitting the next state, A and B registers in to three separate registers. The splitting alone is not enough, since all three would still update at all cycles, so an analysis into the combinatorial part that decide which register needs to be updated needs to be done. To direct the data into the right register could be done with a de-multiplexer, found on page 76 in [6]. and explained in 3 The request out signals of controller A and B must be Multiplexed together for correct behavior, Desynchronization of the more fine grained GCD can be seen in fig: 5.5. The behaviorial of both asynchronous versions are the same although processing time is different in the three. see fig: fig:gcdsimuleringer

5.1.1 Post synthesis measurements

The VCD files of all three versions of the GCD (synchronous, first asynchronous, fine grained asynchronous) have been analyzed using the matlab script, the area estimation from the synthesis tool are listed below, The percentage increase in area is calculated from synchronous to the area of the fine grained. There is a huge increase in non combinatorial logic, this must be caused by the splitting of the registers. Interestingly the extra multiplexer, the handshake demux and mux, plus the extra fork and join only causes and increase of 25% in the combinatorial logic. The results show a decrease in spikelevels for both desynchronized circuits, the finegrained version actually show a significant lowering in the spikes, this must be from the one registers and control that is kept idle.

Area Reports GCD				
	Sync	Async	Fine Grained	increase
Combinatorial	890	903	1095	25%
Non Combinatorial	373	764	716	92%
Total Area	1263	1667	1811	43%

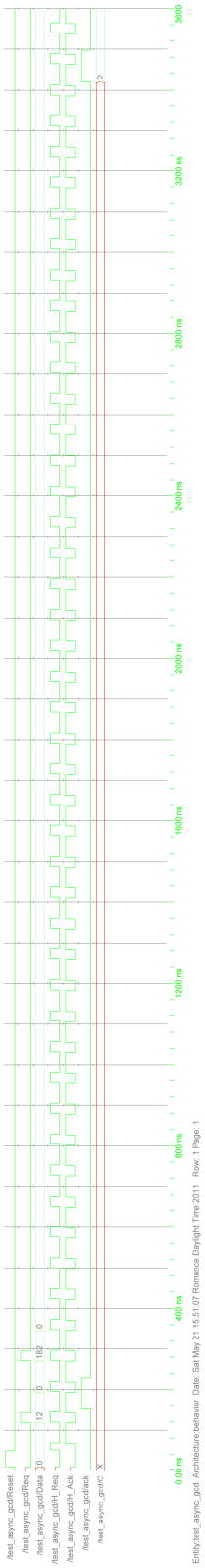
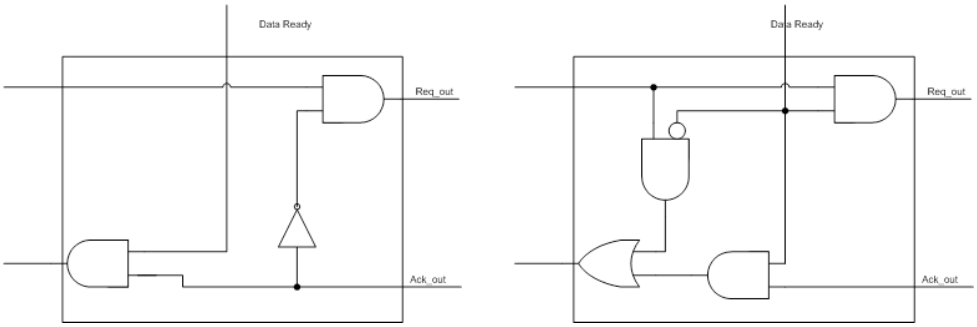


Figure 5.3: The Asynchronous behavioral of GCD



(a) Logic needed at input (b) Logic needed at output

Figure 5.4: Gate implementation of choice logic to stall until calculation has finished.

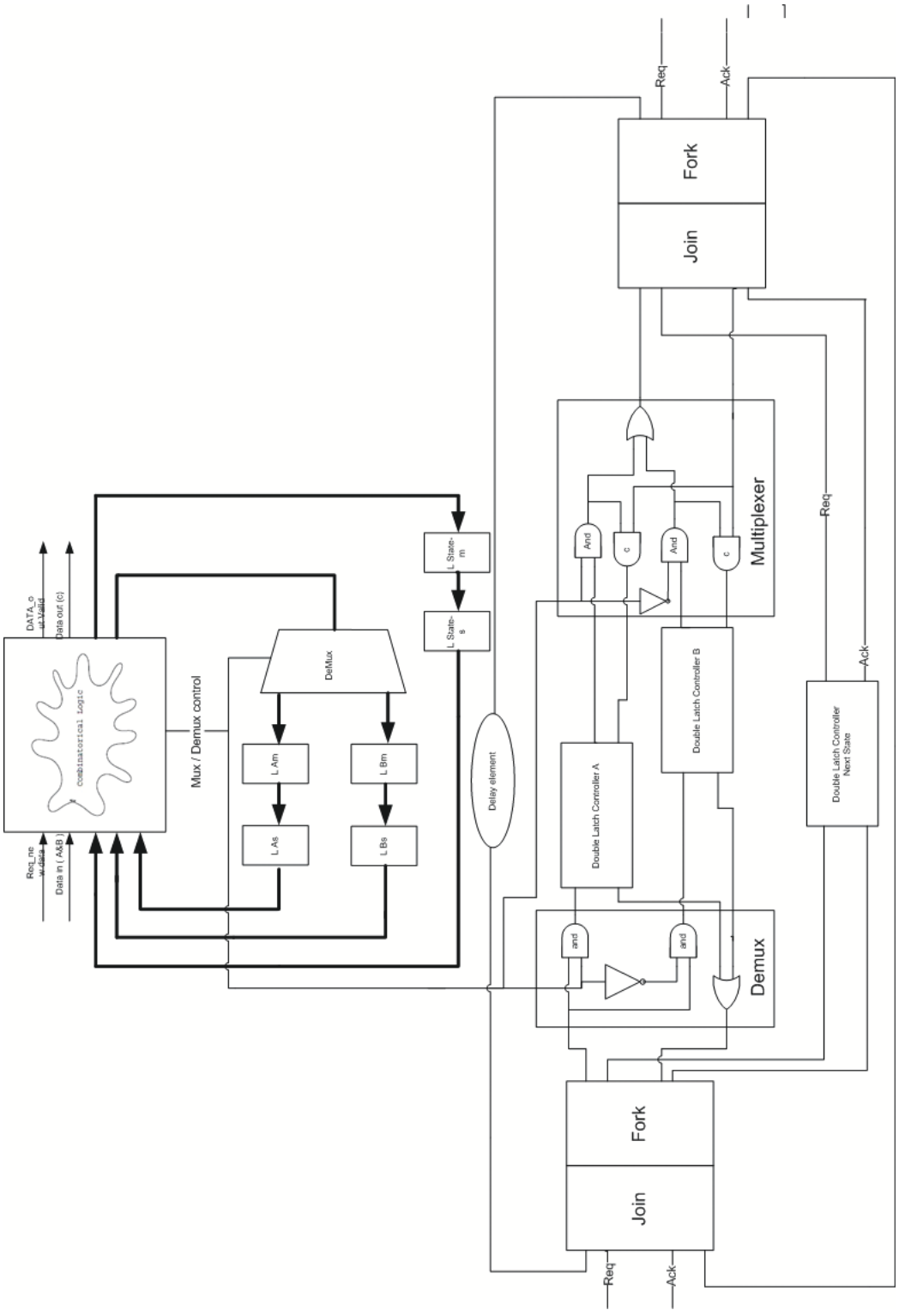
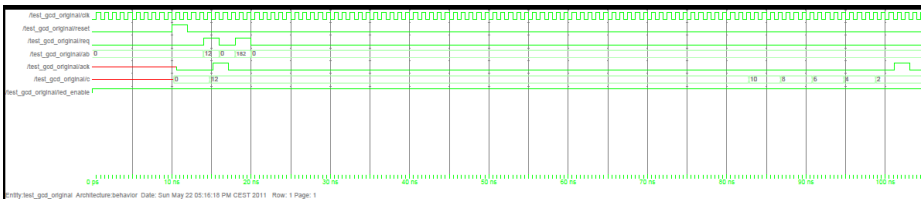
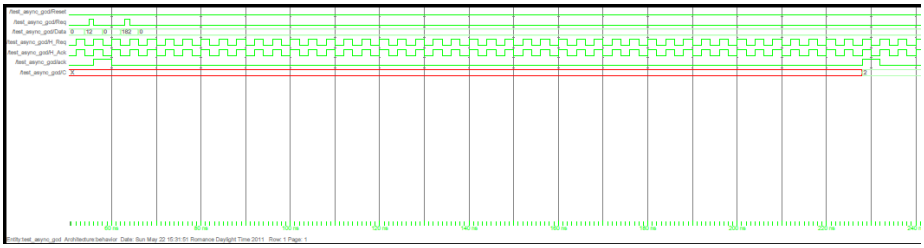


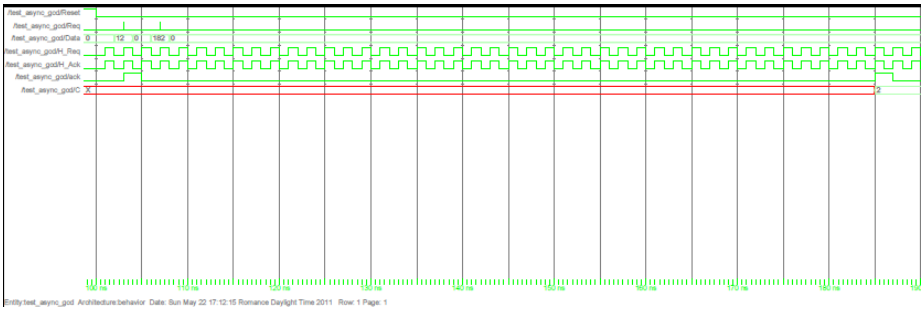
Figure 5.5: A fine grained version of GCD, signals from control to latches are not shown for better overview



(a) Synchronous GCD post synthesis simulation

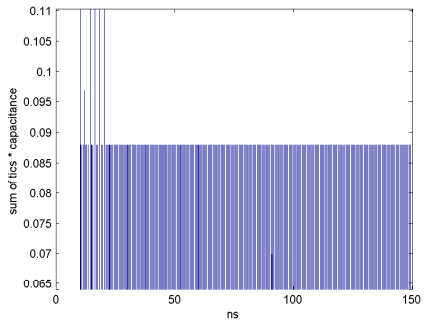


(b) Asynchronous GCD post synthesis simulation

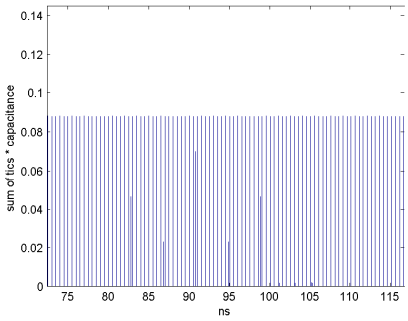


(c) Asynchronous fine grained GCD post synthesis simulation

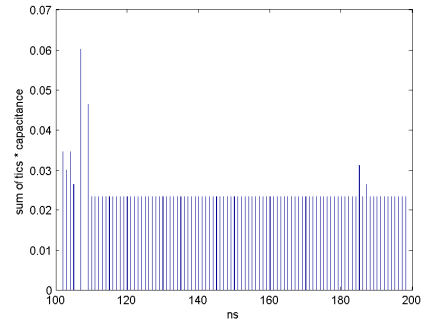
Figure 5.6: Modelsim simulations of the three synthesized implementations



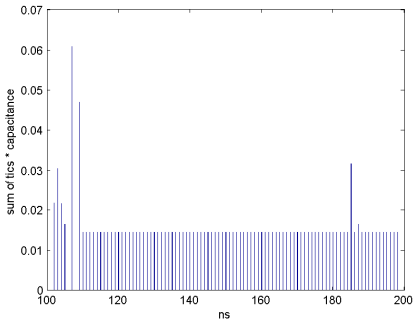
(a) Matlab analysis of the synchronous GCD simulation



(b) Matlab analysis of the synchronous GCD Zoom



(c) Asynchronous GCD post synthesis simulation



(d) Asynchronous fine grained GCD post synthesis simulation

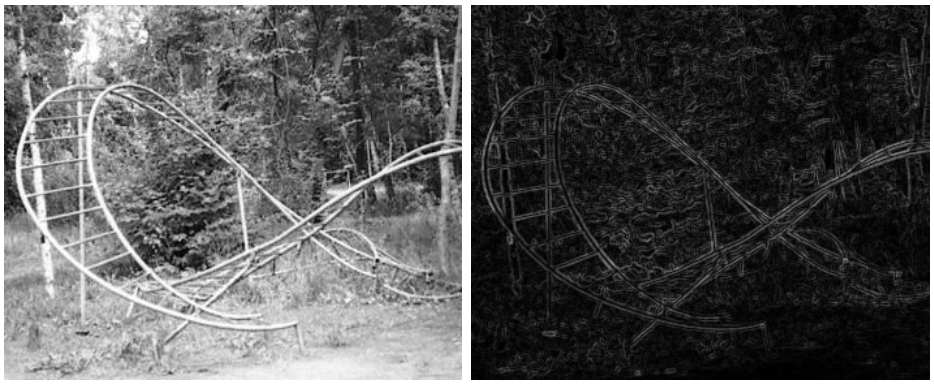
Figure 5.7: Modelsim simulations of the three synthesized implementations

Chapter 6

Example 3 - Edge detector

6.1 Synchronous Edge

The edge detector is taken from a project done in a course at DTU. The project was to develop and implement a system that detects edges in a monochrome intermediate format (CIF) image. The system is designed to work as a hardware accelerator on a bus shared by a CPU. The design is divided into two main blocks a datapath and a Finite state machine. The system receives a start signal from the CPU, and after the image has been processed it raises a finish signal. The actual edge detection algorithm implemented is the Sobel algorithm which is a convolution filter that when applied both horizontally and vertically detects the edges in an image. The filter uses 9 input pixels for every pixel processed. These 9 pixel are previously read from the memory into some input registers. Each read is 32 bits (4 pixels of 8 bit each). The Read and write protocol of the edge detector is such that when the start signal arrives there is 6 reads to buffer pixels, after these 6 reads, the calculation starts. While a pixel is being processed the next four is read in from the memory, every time four pixels have been processed they are saved in the memory. Besides the 6 buffering stages, a pixel is processed at every clock cycle until the total image has been processed. One of the difficulties in the problem was how to deal with edges, the algorithm implemented used even mapping which basically means that the pixels at the edges are re-used to get all 9 pixels needed for processing one. An example of a



(a) Cif image to be processed (b) Cif image after processing by edgedetector

Figure 6.1: The image before and after edge detection

Cif image and the result of the edge detection algorithm can be seen in 6.1, the block diagram of the edge detection unit can be found in fig:6.2.

Finite state machine The finite state machine consist of 5 counters and the state machine it self, the State machine and the counters produce all the control signals for the edge detection unit, when to read and write from memory and at what address, which input, and output register to store the pixels in, which 9 of the buffered pixels to use etc.

Datapath The datapath handles the data flow. There is 9 32bit (4 pixels) input registers capable of storing 36 pixel at the same time, at every clock cycle 9 out of the 36 is the input of the vertical and horizontal filters, the selection of which nine is done by a multiplexer. After the filters the processed pixel is stored in an output register that can store up to 4 pixels (32bit) when the output register is full the 4 pixels are written to the memory. The output register is capable of storing a pixel and writing to the memory in the same clock cycle. This was done to avoid stalling the circuit every time pixels was written to the memory.

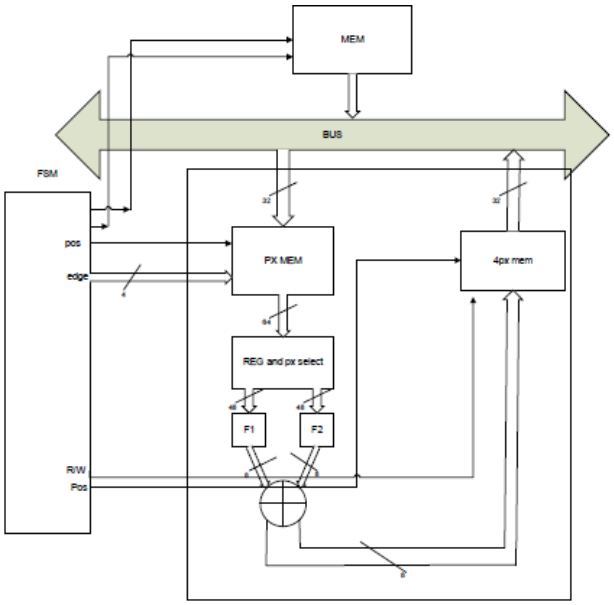


Figure 6.2: Block diagram Edge detector

6.1.1 Desynchronization of Finite state machine

To desynchronize the edge detector it is necessary to dissolve the system into registers and combinatorial circuitry. Since this project was not made with the thought of desynchronization, the VHDL is not formatted in an desynchronization friendly way, so the first thing to do is take out all registers, such that the only thing left is combinatorial blocks. Starting with the FSM block, Schematic is shown in fig: 6.4 The FSM it self is a two process design just like the GCD, so the register is easy to substitute, but the counters used are not formatted in the same way, the VHDL code of a very simple counter is listed here:

```
1  counter : Process(Reset, clk)
2          begin
3                  if( clk'event and clk ='1') then
4                          if(Reset = '1') then
5                                  temp_count <= "0000";
6                          else
7                                  temp_count <= temp_count+1;
8                          end if;
9                  end if;
10 end process;
11 count <= temp_count;
```

The register is not easily detectable, but from the RTL schematic fig: 6.3 of the above code the register is easy to locate, and by using the RTL the counter is recoded into a two process component for easy register extraction. After all registers have been extracted fig: 6.5 the desynchronization of the FSM can begin.

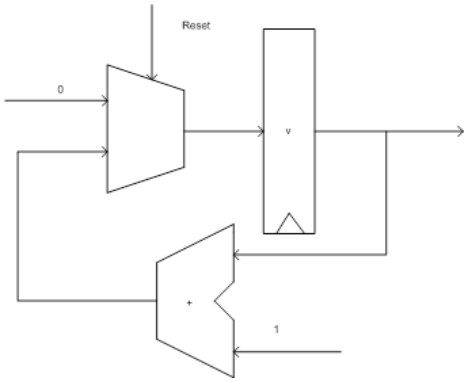


Figure 6.3: RTL diagram of a simple synchronous counter

6.1.2 Inserting latches, and locating forks and joins

The first step is to replace all registers with a double latch + controller, this is simple after the re-coding done in the previous step. The next step is to follow the data and insert a for every time data splits into multiple registers and to insert a join every time a component receives multiple inputs. The FSM delivers data out of the component, and also to every one of the counters there are 5 counters + the data out and input to the combinatorial FSM block, which means a fork of 1:7 is needed, all counters receive data from the FSM block, but also the previous count serves as input, therefore a join is needed in front of every counter control. Four of the counters delivers data out of the FSM top component and also input to the counter it self, so here a fork of 1:2 is needed. Some of the counters also delivers data back to the FSM block, this connect does not need handshaking, when a request from the FSM block is a request to calculate the value needed in the next handshake sequence, and the acknowledge signal is not from the counter is not sent before the data is saved. The time to calculate the new value requires a delay, but this delay can be placed on the acknowledge wire. The Desynchronized FSM is shown in 6.6. The Behavior is verified via simulation, the simulation is shown in 6.7

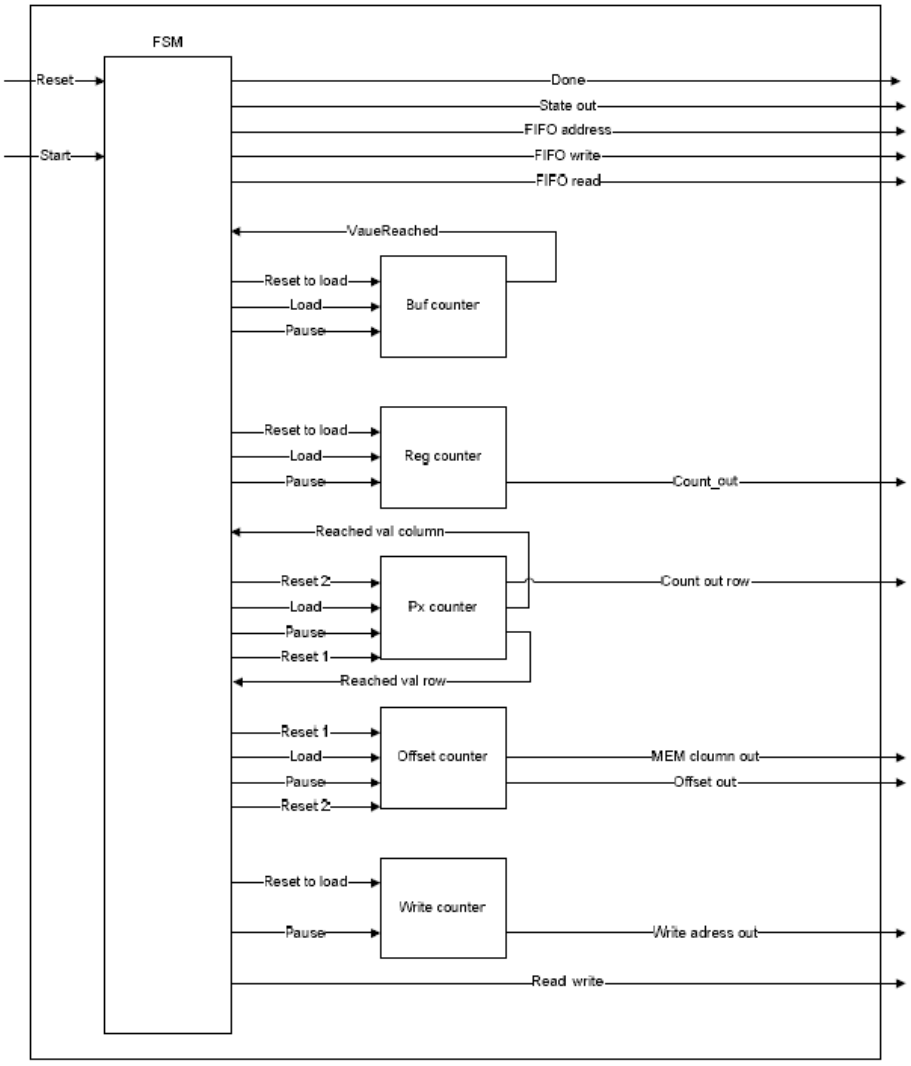


Figure 6.4: Block diagram Finite state machine

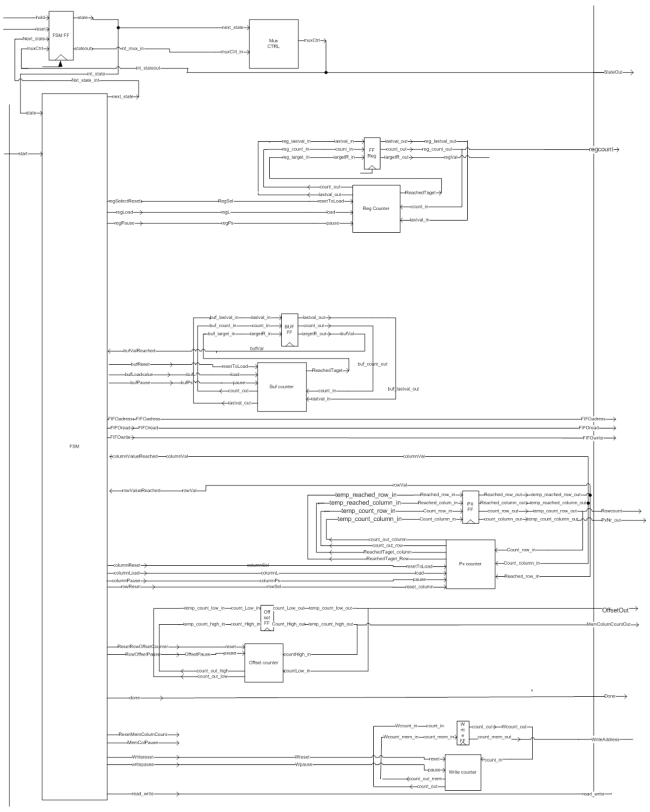


Figure 6.5: Block diagram after register extraction of the Finite state machine

6.1.3 Desynchronization of Datapath

The Data path consist of nine 32 bit input registers that are used to store the reads from the memory, after the input registers are the edge detection algorithm and the result is stored an output register, that can read and write in the same clock cycle. Reading and writing in the same clock cycle is not a problem after desynchronization, since there is no global clock, hence no global cycle time, the circuit will simply be stalled until the output have been saved in the memory.

Desynchronization of the input register (PxMem) the input register is called PxMem short for pixel memory, it stores the pixels read from the memory until they are needed for processing. The registers is already in individual

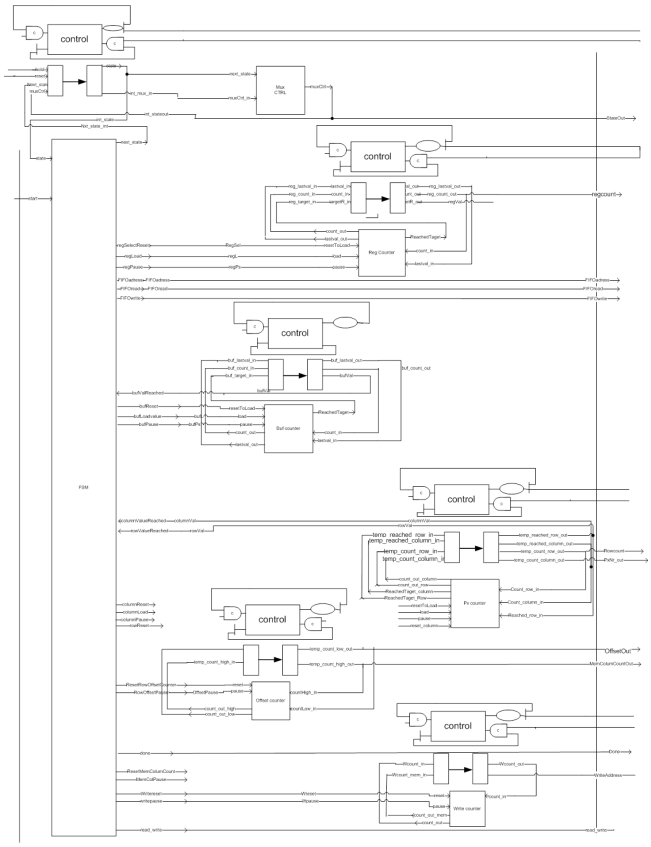


Figure 6.6: Block diagram of the desynchronized Finite state machine, handshake signals between FSM control and counters not shown. All counters handshake with FSM

components, so there is no need for re-coding this block. There is a total of nine 32 bit registers, but only one is loaded with a new value at every read. The nine registers are collected into groups of three, each group is a subcomponent of the PxMem. To be able to only activate one or none of the registers in each block a de-multiplexer is needed. There is already implemented a 2:3 decoder in each block these 3 control signal currently used as a register enable, can be used as control signals in the de-multiplexer. Also a multiplexer for selecting the correct request signal from the registers is needed. Both components are shown in fig: 6.8. Only one register out of the nine is active at any given time, therefore the same de-mux and mux can be used to select which of the three identical block should receive the in put request. Each block receives 2 bit signal indicating

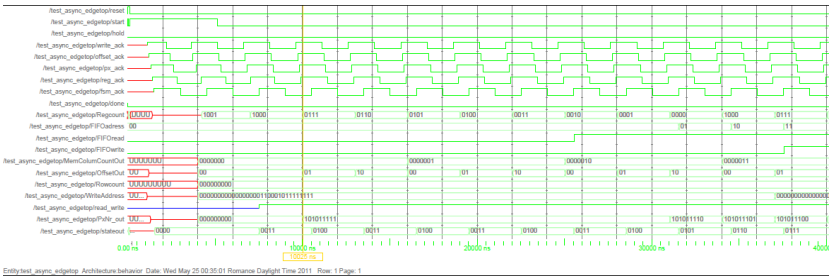
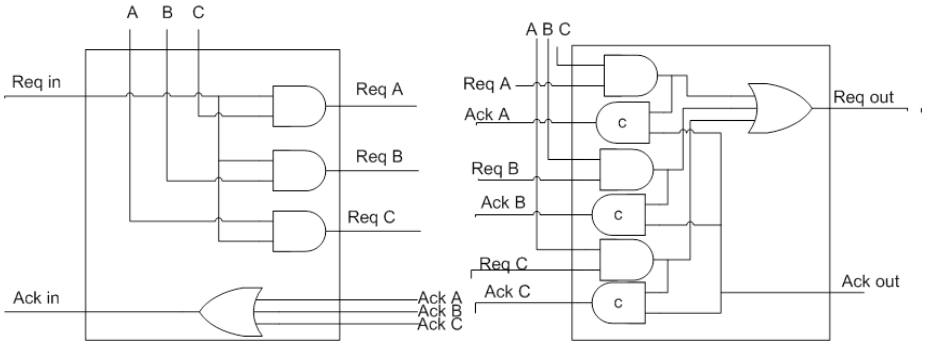


Figure 6.7: Simulation of the Asynchronous FSM

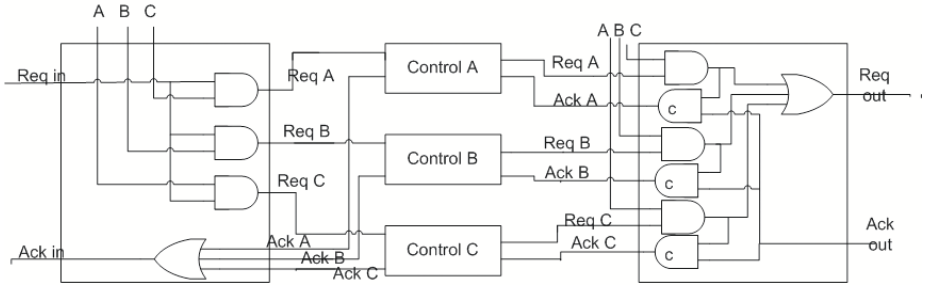
which registers are active "00" means all three registers in the block are inactive, to get the control signals of the mux and the de-mux simply OR the two control signals. Only one delay element is needed since only one register is active at the time, the element should be placed after the multiplexer and before request out of PxMem. The schematic complete desynchronization of the PxMem can be found in fig: 6.10

Desynchronization of the output register (savePxl) The savepxl component is actually two separate registers. The pxl2bus is a simple 32bit register which is very simple to dezynchronize, and the rest of the savepixel component is a multiplexer and four 8bit (1 pixel) registers, according to the address the input is saved into one of the four registers. When desynchronizing this part there is two possibilities, either to desynchronize the four pixel registers as one big, or as four separate. choosing the latter has more overhead since 4 sets of controllers is needed, but there is a gain since three out of four of the registers will be idle. and will therefor not contribute to power consumption. The latter is chosen. As in desynchronization of the PxMem a de-mux and a Mux is needed, this time its a 1:4 de-mux and 4:1 mux.

Inserting forks, joins and delay elements After desynchronization of all the register components in the datapath it is time to connect them. The datapath receives data from several counters and from the FSM block therefor some joins are needed to synchronize the results, the output is only to the bus so only nor forks is needed in this component. The desynchronized version of the datapath is shown in fig: 6.11



(a) De-mux used in datapath (b) Mux used in datapath



(c) Handshake circuit of PxMem, latches not shown!

Figure 6.8: 1:3 De-multiplexer and 3:1 multiplexer

6.1.4 Connecting the FSM and the Datapath

The only thing left to do is to connect the two top components. There are is a read/write decoder component also, this is purely combinatorial but some delay is needed to simulated the delay that the data control signals experience through this. The start signal that indicates that a new images is ready for processing should be held long enough for a complete handshake sequence to take place to ensure that the process actually starts.

Behavioral model It was not possible to finish a working behavioral of the edge detection component before the deadline of this project. This is due to the fact that debugging of asynchronous signals is very tedious work that often takes days. The theory behind the desynchronization and the schematics presented should although be correct since the subcomponents have been tested and found error free. The problem could be in several places:

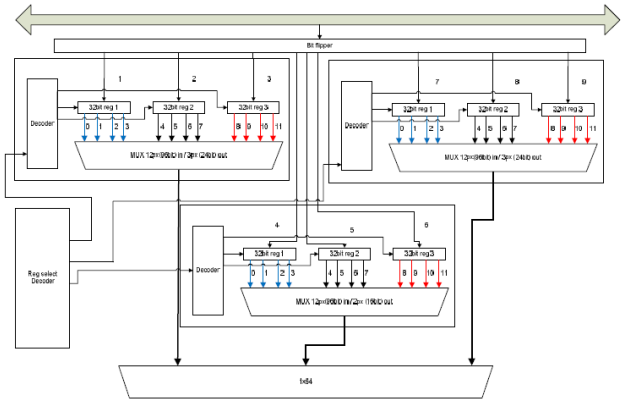


Figure 6.9: The RTL of the input register in the Edge detection datapath.

- Bad wiring, some handshake signals might be connected in a wrong way.
- A delay element may be placed in a wrong place, or might be missing.

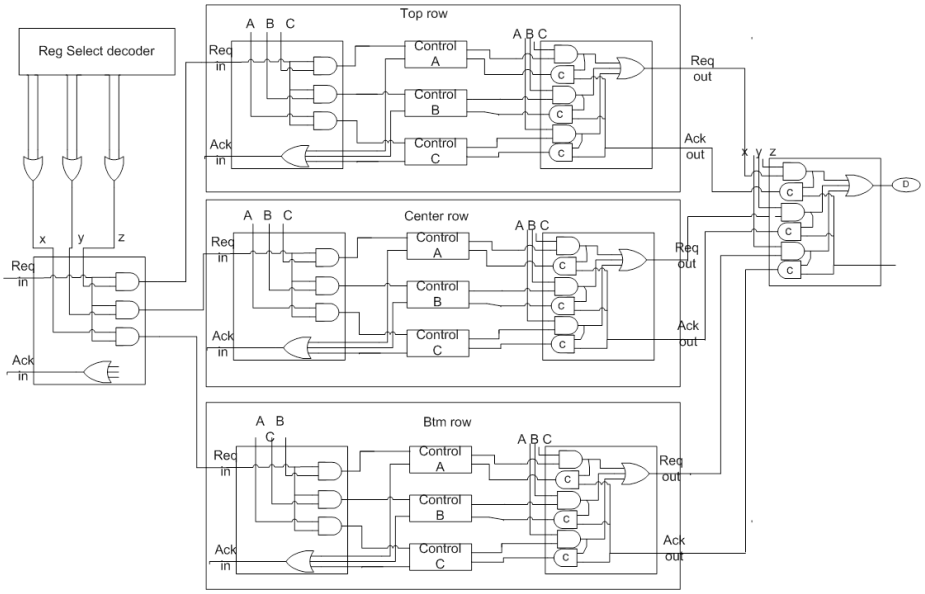


Figure 6.10: Schematic of PxMem after desynchronization, only handshake logic is shown

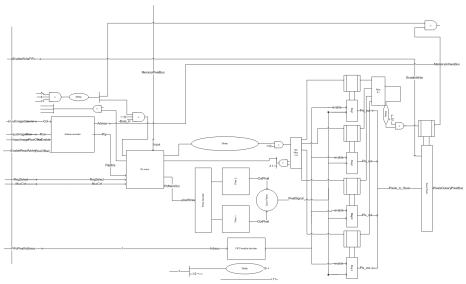


Figure 6.11: Schematic of desynchronized datapath, wire ends are numbered to indicated connections

Chapter 7

Discussion

In this chapter the outcome of the project is discussed and evaluated, also some improvements are proposed together with ideas for future work.

7.0.5 Evaluation

There were two primary goals for this thesis, one was to develop a design flow for desynchronization, the other was to try to prove some of the theoretical advantages of desynchronization. The design flow was established, and the first two examples show that it is indeed possible to desynchronize a synchronous specification. The third example Edge detection algorithm is presented as a schematic and the steps to desynchronize the component is described in detail although a working behavioral was not reached, the FSM component and the datapath was desynchronized with success and works as separate components. The results obtained did not show give any solid reason to conclude that desynchronized circuits hold any advantages over the synchronous, there are indications that the average spikes of the switching activity is lowered. The reason that no conclusive result where obtained is a combination of two.

- The task of learning the synthesis tool Synopsys was a big challenge, not only to learn basic synthesis but the job of tweaking the compiler to be

able to compile and synthesize the asynchronous component, for which the tool was not designed for, proved to be more difficult than expected. The use of synopsis was not seen as a part of the project but rather as a method to evaluate and validate the desynchronized designs. but in the end the tweaking and use of Synopsys for asynchronous systems could probably fill a thesis project by itself.

- The two working examples are very small, and only uses one or two registers and controllers. Both of them only incorporates a single matched delay. The use of matched delays indicates a fixed timing, and therefore some what of a synchronization of the events. A synchronous pipeline have clock ticks evenly spaced over time, the asynchronous pipeline may not have evenly spaces ticks, but with the use of delay elements the tick of each stage will arrive at the exact same time relative to the ticks of the previous pipeline stage, so the use of matched delay elements in pipelines will have a tendency to give a synchronous like result. In highly parallel designs the matched delay element could prove to be a good way of distribute the switching out in time. This theory was not tested in this thesis. another method of distributing the clock tics could be another form of completion detection, or completion prediction.

The desynchronization of the greatest common divisor indicates that the area overhead of fine grained desynchronization is small and it also shows a decrease in the switching spikes, but this test was done with an average capacitance for a lot of internal nodes due to problems with the synthesis tool reporting. Robustness towards temperature and voltage variations is not tested, so no conclusion about the performance in tracking the combinatorial can be made. One of the problems with Synopsis is the timing is based on a clock, so with out a clock Synopsys does not provide any information about the timing in the combinatorial circuits. A method suggested in [4] is to generate virtual clocks to simulate the ticks of the handshake circuit.

7.0.6 Future work

Some future work could be:

Delay tracking It could be interesting to see how the matched delay tracks the delay in the combinatorial path, the desynchronized circuit may hold some advantages over the synchronous one in this matter.

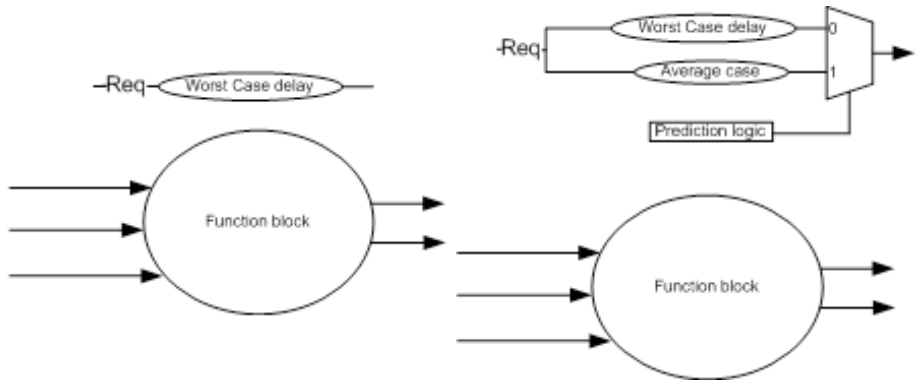
Synthesis tool A better understanding of the synthesis tool, and how to use it for asynchronous design, could very well give better test results, a possibility could be to use Nanosim for the transient simulations. This should give a very precise image of the switching activity and and the instantaneous power consumption.

Completion prediction and completion detection In the desynchronization method proposed the delay through the combinatorial paths of the circuit is mimicked by a matched delay element, this delay is constant and must be designed for the worst case scenario. In reality the general calculation time is less, and maybe a lot less than the worst case design, at the same time the constant delay also has a tendency to synchronize the timing in the circuit. An improvement could be either to implement completion detection or completion prediction.

The handshake protocol in this thesis is a four-phase bundled data protocol. Another method could be to implement the dual-rail protocol. This has the completion detection built into the data wires, and therefore there is no need for a delay element. This would insure that the delay is always matched to the actual calculation time, and the delay would automatically be adjusted for the calculation done. The cost of implementing the dual-rail is area overhead, which could prove to be significant.

Another solution could be completion prediction. Some combinatorial logic is inserted to evaluate the input data, and based on this evaluation a delay is chosen. The evaluation logic could control a simple multiplexer that selects one of its inputs. The two implementations is shown in fig: 7.1 A good example of this is in [8] where a carry prediction circuit is implemented to select the delay through a Brent-Kung adder. The prediction evaluates how long the carry chain in the adder will be. The selection is done between three different delay elements, one for worstcase delay and two for speculative completion. This method is less complex than the dual-rail structure and could give a significant performance increase.

Granularity An investigation into how fine grained desynchronization should go before the area overhead is too much compared to the gain in power consumption.



(a) The Matched delay implementation used in this thesis

(b) The Matched delay implementation used in this thesis

Figure 7.1: Regular delay matching and Predictive delay matching

Chapter 8

Conclusion

The difficulties in prediction and tracking variations become increasingly difficult with the introduction of nm-scale. A solution proposed is to use asynchronous design instead of synchronous since asynchronous circuits in theory hold some advantages over the synchronous equivalents.
This thesis has proposed an easy step by step guide to desynchronize a synchronous circuit, where every step is explained and demonstrated in detail. A design flow are proposed with reasonable success, the results produced by this is inconclusive about the claimed advantages of the asynchronous circuit.

The Step by step guide and synthesis flow were tested on three circuits from very small scale with only one register, to a relatively big Edge detection algorithm using multiple registers with choice components (mulitplexer and de-mulitplexer). The desynchronizing of the edge detection circuit was not completed within the deadline of this project. Also a small guide for general synthesis using Synopsys was developed during this thesis.

The tests show that the use of matched delay elements have a tendency to synchronize the switching activity, and therefore it has little advantage over the synchronous equivalent. The theory about idling was proven, the examples clearly shows that the asynchronous circuits does not have any switching activity until the start signal.

Appendix A

Small Design vision guide

This is short step by step guide on how to synthesize VHDL in synopsis design vision. The VHDL design to be synthesized is the asynchronous adder from the thesis.

A.1 Design vision

Start design vision with the command $design_vision-db$, the db command opens design vision i database mode for easy storing of synthesis reports. When design vision first opens, it will open an empty window. See fig: A.1

To import the VHDL files click file-¿Analyze, in the pop-up window click add. Select all files except the test bench files and click ok. Synopsys need to read the files bottoms-up, find the top files and move it to the bottom of the list using the arrows. fig: A.2.

Click file -¿ elaborate, the window should now show all your components and the hierarchial structure of the design. To see the design click schematic -¿ new schematic. The design vision should now look like fig: A.3. Try to double click the fork you should get something similar to fig A.4, Notice the delay component does not hold any combinatorial logic, it is just a wire.

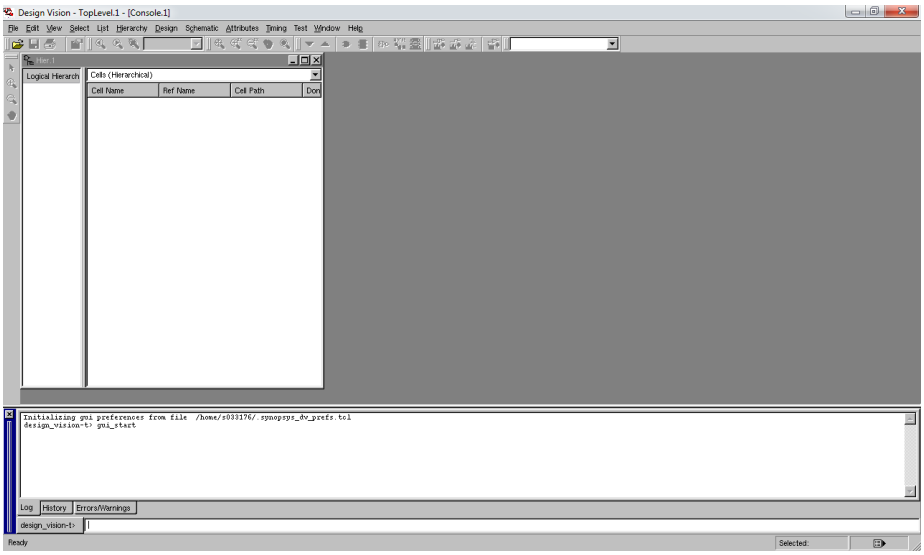


Figure A.1: Design vision when first opened

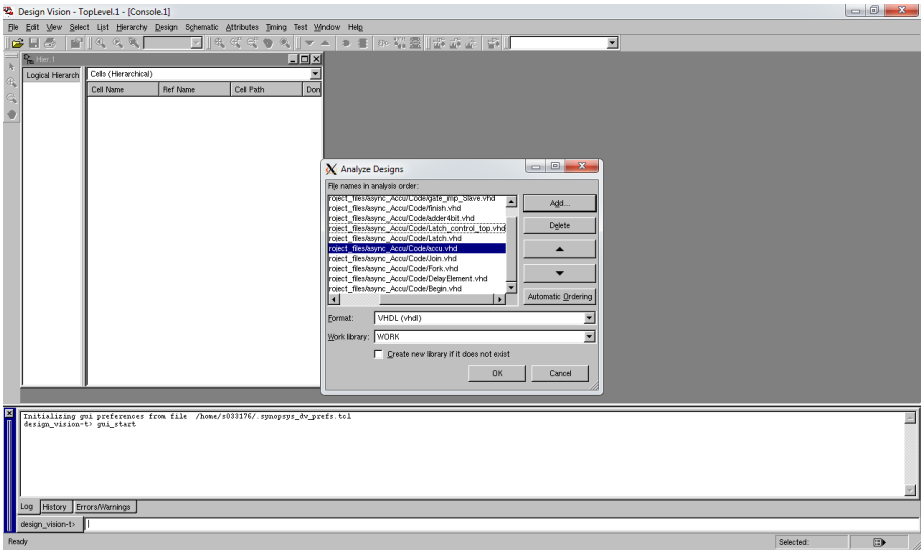


Figure A.2: Design vision when first opened

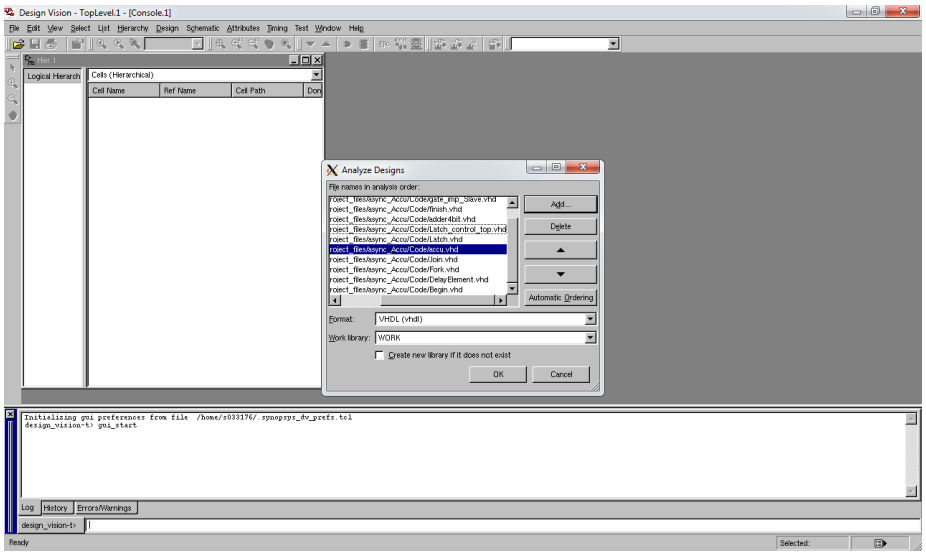


Figure A.3: Design vision when first opened

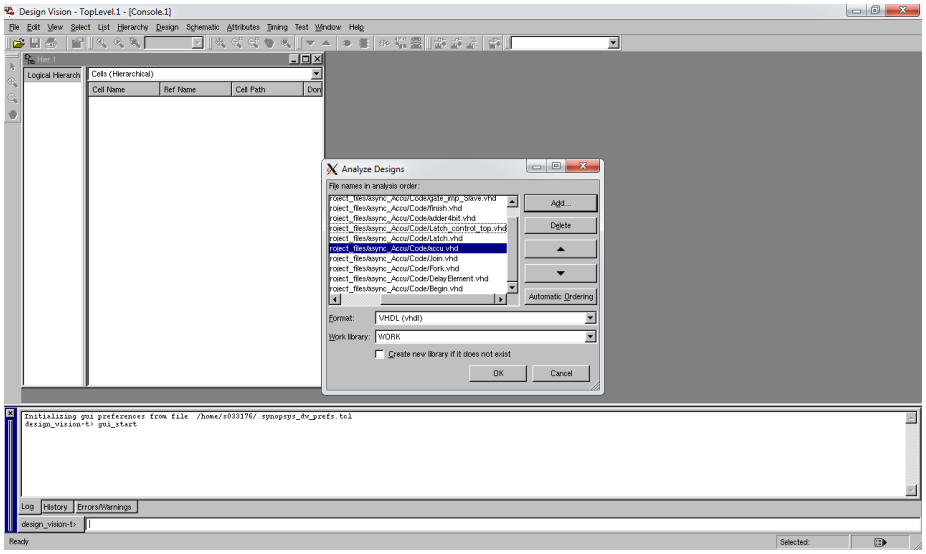


Figure A.4: Design vision when first opened

click compile-¿compile design to synthesize. A pop-up telling you that some schematics has changed. to see the new schematic click schematic -¿ new schematic, in fig:A.5 the fork is shown after synthesis. The C-element in the fork has changed. It is still a valid implementation of the C-element.

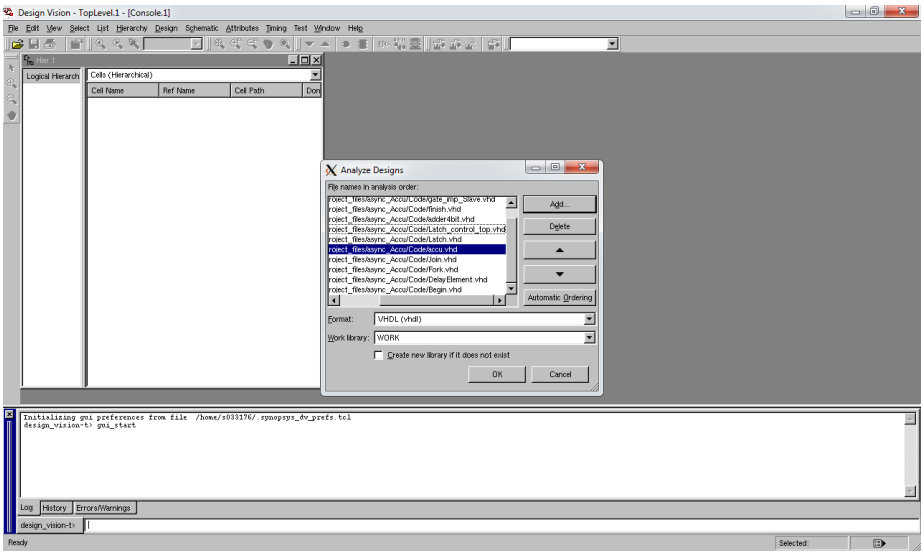


Figure A.5: Design vision when first opened

To save the design to a verilog netlist, click file-Save as and select verilog as the format. the various needed reports can be created unsing report -¿ deired report.

Appendix B

scripts and files for Synopsis and Matlab

B.1 compile script for synopsis synthesis

```
sh date

##############################i
##  Source RTL FILE paths ##
##############################
source ./scripts/rtl.tcl

########################################
##  Setup logic and milkyway libraries  ##
########################################
source ./scripts/setup.tcl

#################
##  READ RTL  ##
#################
set top_module DelayTop
analyze -format vhdl -define RUNDC -lib work ${rtl_files}

elaborate ${top_module} -lib WORK

######################
## POWER ESTIMATION ##
######################

#set power_preserve_rtl_hier_names true
#link
#rtl2saif
```

```
28
29  ################################
30  ##  READ TIMING CONSTRAINTS  ##
31  ################################
32  #set PERIOD 1.0
33  #create_clock -name "CLK" -period $PERIOD [get_ports clk]
34
35  current_design ${top_module}
36  uniquify
37  link
38
39  ############################
40  ##  Source Power domains ##
41  ############################
42
43  # Top power domain
44
45  create_power_domain TOP
46
47  # Create power net info
48
49  create_power_net_info -power VDD
50
51  create_power_net_info -gnd VSS
52
53  # Power net info for top module
54
55  connect_power_domain TOP \
56                       -primary_power_net VDD \
57                       -primary_ground_net VSS
58
59  report_power_domain > ./report/power_domains
60
61
62  set_operating_conditions -max NomLeak -min NomLeak
63  ########################
64  ##      Compile       ##
65  ########################
66  current_design ${top_module}
67  compile
68
69  #################
70  ##  DATA OUT   ##
71  #################
72  change_names -rule verilog -hier
73  write -f verilog -h -out ./db/${top_module}_postsyn.v
74  write -f ddc -h -out ./db/${top_module}_postsyn.ddc
75  write_sdc -nosplit ./db/${top_module}_postsyn.sdc
76  write_link -nosplit -out ./db/${top_module}_postsyn.link
77  set mw_design_library ./MW_FPMULT
78  write_milkyway -out compile -over
79
80  ## REPORT
81  #########
82  check_mv_design -verbose > ./report/mv_check_compile.rpt
83
84  report_cell > ./report/${top_module}_postsyn_cells.rpt
85  report_area > ./report/${top_module}_postsyn_area.rpt
86  report_net  > ./report/${top_module}_postsyn_net.rpt
87
88  #report_hier -nosplit -noleaf
89
90  report_timing -att \
91                -net \
92                -trans \
```

```
93                -cap \
94                -input \
95                -volt \
96                -nosplit > ./report/${top_module}_postsyn_tim.rpt
97
98  report_power -hier -hier_level 1 -verb > ./report/${top_module}
        _postsyn_power.rpt
99
100 sh date
101
102 #exit
```

B.2 floorplan script for synopsis

```
1   sh date
2
3   ########################################
4   ##  Setup logic and milkyway libraries  ##
5   ########################################
6
7     source ./scripts/setup.tcl
8
9   ######################
10  ##  open dft cell  ##
11  ######################
12
13    copy_mw_cel -from compile -to floorplan
14    open_mw_cel floorplan
15    link
16    link_physical
17
18  ############################
19  ##  Floorplan generation ##
20  ############################
21
22    initialize_floorplan -control_type  aspect_ratio  \
23                         -core_aspect_ratio 1.0 \
24                                  -core_utilization 0.7 \
25                         -left_io2core 10 \
26                         -bottom_io2core 10 \
27                         -right_io2core 10 \
28                         -top_io2core 10
29
30  derive_pg_connection -power_net VDD -ground_net VSS
31
32  #################
33  ##  Placement  ##
34  #################
35
36   create_fp_placement
37
38  ###############################
39  ##  Power Network Synthesis  ##
40  ###############################
41  create_rectangular_rings  -nets  {VDD VSS} \
42                            -left_offset 1 -left_segment_layer M4
                                  -left_segment_width 2 -extend_ll -extend_lh \
43                            -right_offset 1 -right_segment_layer M4
                                  -right_segment_width 2 -extend_rl -extend_rh \
```

```
                          -bottom_offset 1 -bottom_segment_layer M3
                               -bottom_segment_width 2 -extend_bl -extend_bh
                               \
                          -top_offset 1 -top_segment_layer M3
                               -top_segment_width 2 -extend_tl -extend_th
                               -offsets absolute

create_power_straps  -nets  {VDD} \
                     -layer M4 \
                     -width 2 \
                     -direction vertical \
                     -start_at 31.2 \
                     -num_placement_strap 2 \
                     -increment_x_or_y 147



create_preroute_vias  -nets  {VDD} \
                      -from_layer M4 \
                      -from_object_strap \
                      -to_object_std_pin_connection \
                      -to_object_std_pin \
                      -within {{-22.235 -4.945} {217.200 220.495}}


set_fp_rail_constraints  -skip_ring -extend_strap core_ring
set_fp_rail_constraints -add_layer  -layer M6 -direction horizontal
    -max_strap 32 -min_strap 1 -min_width 0.2 -spacing minimum
set_fp_rail_constraints -add_layer  -layer M5 -direction vertical -max_strap
    32 -min_strap 1 -min_width 0.2 -spacing minimum
################
##  PNS VSS  ##
################

synthesize_fp_rail  -nets {VSS} -power_budget 2 -voltage_supply 1.0
    -use_pins_as_pads

commit_fp_rail

###############
## PNS VDD  ##
###############

#set_fp_rail_voltage_area_constraints -net  {VDD} \
#                                     -power_budget 1 \
#                                     -voltage_supply 1.0

synthesize_fp_rail -use_pins_as_pads -nets {VDD} -target_voltage_drop 200.0
commit_fp_rail

preroute_standard_cells -nets  {VDD VSS}  -connect horizontal   \
                                          -port_filter_mode off
                                              -cell_master_filter_mode off \
                                          -cell_instance_filter_mode off \
                                          -voltage_area_filter_mode select

write -f verilog -h -out ./db/floorplan.v

remove_stdcell_filler -stdcell

save_mw_cel -increase_version

close_mw_cel

sh date
```

```
101
102 exit
```

B.3 matlab file for comparing switching activity

```
1  clear all
2  %close all
3  %% inputs
4  clk = 1000; % in ns Rethink this
5  ResetName = 'Reset';
6  %fileName = 'C:\Rasmus\My Dropbox\Speciale\Matlab\Accu\Sync_accu_post_syn.vcd
       ';
7  %fileName = 'C:\Rasmus\My Dropbox\Speciale\Matlab\Accu\Async_Bool_noSynth.vcd
       ';
8  %fileName = 'C:\Rasmus\My Dropbox\Speciale\Matlab\Accu\Async_accu_req.vcd';
9  %%%%%%%%%%%%%%%%%% ACCU %%%%%%%%%%%%%%%%%%
10 %sync ACCU
11 %fileName = 'C:\Rasmus\My Dropbox\Speciale\Projects\Accu\Vhdl_sync\VCD files\
       sync_accu_synth.vcd';
12 %ASync_ACCU
13 %fileName = 'C:\Rasmus\My Dropbox\Speciale\Projects\Accu\Vhdl_async\Syntesize
        result\Async_accu_synth.vcd';
14
15 %%%% GCD %%%%
16 %fileName = 'C:\Rasmus\My Dropbox\Speciale\Projects\GCD\modelsim_postsynth\
       GCD_sync_post.vcd'; %sync
17 %fileName = 'C:\Rasmus\My Dropbox\Speciale\Projects\GCD\modelsim_postsynth\
       GCD_Async_post.vcd'; %async Reset Med stort R
18 fileName = 'C:\Rasmus\My Dropbox\Speciale\Projects\GCD\modelsim_postsynth\
       GCD_FG_async_post.vcd'; %async Reset Med stort R
19
20 %%
21 clc
22 fid = fopen(fileName,'r');
23 filedata=textscan(fid,'%s', 'delimiter', '\n');
24 data = filedata{1};
25 fprintf(' File loaded \n');
26
27 fclose(fid);
28
29
30 %% find timescale
31 fprintf(' Find Timescale \n');
32 [TT]= regexp(data,'timescale', 'match');
33 Row = find(~cellfun(@isempty,TT)) ;
34 str1 = regexp(data(Row+1), 'p','split');
35 str2 = str1{1,1};
36 Timescales = str2num(str2{1});
37 %% find Reset
38 fprintf(' locate reset symbol and first reset high \n');
39 [TT]= regexp(data,ResetName, 'match');
40 Row = ~cellfun(@isempty,TT) ;
41 str1 = regexp(data(Row), ' ','split');
42 str2 = str1{1,1};
43 RST = str2(find(strcmp(ResetName,str2)==1)-1);
44 RSTfirst = strcat('1',RST);
45
46 %% find wire names and symbols and loads
47 fprintf('find wire names and symbols\n');
```

```
[TT]= regexp(data,'wire', 'match');
Row = ~cellfun(@isempty,TT) ;
str3 = regexp(data(Row), ' ','split');
DataNames = struct('Name',[],'symbol',[],'value',[]);
%Sync Accu
%DataNames.value
     ={0.00378700000000000;0.00378700000000000;0.00378700000000000;0.00378700000000000;0.00378700000000000;0.00378700000000000
%Async accu
%DataNames.value
     ={0.00335700000000000;0.00335700000000000;0.00335700000000000;0.00335700000000000;0.00335700000000000;0.00399600000000000
%sync GCD
%DataNames.value
     ={0.0879000000000000;0.00879800000000000;0.00697800000000000;0.00691700000000000;0.00691800000000000;0.00691900000000000
%async GCD
%DataNames.value
     ={0.0248400000000000;0.0126210000000000;0.0298400000000000;0.0298400000000000;0.0298400000000000;0.0298400000000000;0.029
%fine grained GCD
DataNames.value
     ={0.0387150000000000;0.00694700000000000;0.00353400000000000;0.00353400000000000;0.00353400000000000;0.00353400000000000
for i=1:length(str3)
    str4 = str3{i,1};
    DataNames.Name{i,1}=char(str4(5));
    DataNames.symbol{i,1}=char(str4(4));
    %DataNames.value{i,1} = 1; % temp string that assigns all wires same
        capacitance
end
 %DataNames.value{17} = 1; % make the clk expensive
%% find changes in activity
fprintf('find all signal changes');
p = '#(?<Time>[^ \n]*)\n% [^%]*|#(?<Time>[^ \n]*)';

names = regexp(data,p,'names');
Row = find(~cellfun(@isempty,names)) ;


%% Extract values
fprintf('extract timing values \n');
x= length(Row);
activity(x,2) = 0; %generate array for fast calculation
for i=1:x
y =  str2num(names{Row(i),1}.Time);
if isempty(y)
    y=NaN;
end
 activity(i,:) = [Row(i) y/10]; %timescale to ns (100ps means all times must
     be mulitplies by 100 to get time in ps..(devide by 10 to get ns)
end
%% find activity in each section
fprintf('calculate number of changes in interval \n');

%interval = clk/100; % time increment for each interval
% nothing might happen in several concecutive intervals, so for faster
% algorithm calculation, seach for next time something happens and assume
% everyting happens within first section of that interval and nothing
% happens the rest.

%find the first reset event, dont count anything before reset, since we
%have no clue of the behaviour before this point
notEmpty = regexp(data,RSTfirst,'match');

```

```
101 firstReset = find(~cellfun(@isempty,notEmpty)) ;
102
103 % find first event after reset (maybe change to finde reset low)
104 %place = find(activity(:,1)>firstReset(1),1,'first'); % expression , number
        of returns, first or last'
105
106 place = find(activity(:,1)>73,1,'first');
107 %place = find(activity(:,1)>262,1,'first');
108 % if (activity(place,2)== NaN )
109 %     place = find(activity(:,1)>activity(place,2),1,'first');
110 % end
111
112 c1 = activity(place,1);
113
114 %i=place+1; % for test
115  x = length(activity);
116  z=1;
117  for i=place+1:x %
118      if (isnan(activity(i,2)))
119
120      else
121          val=0;
122          c2 = activity(i,1); %end point of interval
123          for j=(c1+1):(c2-1) %find symbol and multiply with capacitance
124              symb = char(data(j));
125              temp_val = DataNames.value{find(~cellfun(@isempty,strfind(
                     DataNames.symbol,symb(2))))};
126              val = val + temp_val(1);
127          end
128          counts(z,1) = val;
129
130          counts(z,2)=activity(i-1,2); %time of the count
131          z=z+1;
132          c1 = c2 ;
133      end
134  end
135
136  %% add activity within same interval.
137  simEnd = activity(x-1,2); %last noted time in simulation (this might need to
         be added manually to the VCD file)
138  NrPoints = clk*100;
139  timesVctr = linspace(activity(place,2),simEnd,NrPoints);
140  intBeg =1;
141  i=2;
142  for i=2:length(timesVctr)
143     intEnd = find(counts(:,2)<timesVctr(i),1,'last');
144     intv(i-1) = sum(counts(intBeg:intEnd,1));
145     intBeg = intEnd+1;
146  end
147  intv(i)=0;
148  total_Ticks = sum(counts(:,1));
149  checksum=sum(intv);
150  figure
151  %if (checksum==total_Ticks)
152  intv=intv.*2.1;
153      bar(timesVctr,intv)
154      xlabel('ns');
155      ylabel('sum of tics * capacitance');
156
157      fprintf('processing successfull');
158 % else
159 %     fprintf('processing failed!  \n Number of ticks %d do not match
        checksum %d \n',total_Ticks, checksum);
160 % end
```

Appendix C

VHDL Code

C.1 VHDL for basic components

C.1.1 c element

C.1.2 VHDL for Fork and Join

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    10:28:06 11/18/2010
6  -- Design Name:
7  -- Module Name:    Fork - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
```

```
19 --
       ----------------------------------------------------------------------------------

20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity Fork is
31     Port ( Req_in : in  STD_LOGIC;
32            Req_out : out  STD_LOGIC_VECTOR (1 downto 0);
33            Ack_in : in  STD_LOGIC_VECTOR (1 downto 0);
34            Ack_out : out  STD_LOGIC);
35 end Fork;
36
37 architecture Behavioral of Fork is
38 signal ack_int : std_logic;
39 begin
40         Req_out <= (Req_in,Req_in);
41         ack_int <= (Ack_in(0) and Ack_in(1)) or (ack_int and (Ack_in(0) or
               Ack_in(1)));
42         Ack_out <= ack_int;
43
44 end Behavioral;
```

```
1  --
       ----------------------------------------------------------------------------------

2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    11:15:14 11/18/2010
6  -- Design Name:
7  -- Module Name:    Join - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 --
       ----------------------------------------------------------------------------------

20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity Join is
```

```
31      Port ( Req_in : in  STD_LOGIC_VECTOR (1 downto 0);
32             Req_out : out  STD_LOGIC;
33             Ack_in : in  STD_LOGIC;
34             Ack_out : out  STD_LOGIC_VECTOR (1 downto 0));
35  end Join;
36
37  architecture Behavioral of Join is
38
39  SIGNAL req_int : std_logic;
40  begin
41
42  Ack_out <= (Ack_in, ACk_in);
43
44  Req_int <= (Req_in(0) and Req_in(1)) or (Req_int and (Req_in(0) or Req_in(1))
        );
45
46  Req_out <= Req_int;
47
48  end Behavioral;
```

C.1.3 VHDL for mux and demux

```
1   --
        ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    21:57:12 05/21/2011
6   -- Design Name:
7   -- Module Name:    async_multiplex - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  --
        ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity async_multiplex is
31      Port ( Mux : in  STD_LOGIC;
32             ReqA : in  STD_LOGIC;
33             ReqB : in  STD_LOGIC;
34             Req_out : out  STD_LOGIC;
```

```
35            AckA : out  STD_LOGIC;
36            AckB : out  STD_LOGIC;
37            Ack_out : in  STD_LOGIC);
38  end async_multiplex;
39
40  architecture Behavioral of async_multiplex is
41  signal int_a, int_b, int_ackA, int_ackB :std_logic;
42  begin
43  int_a <= (Mux and reqA);
44  int_b <= (not(Mux) and ReqB);
45
46  Req_out <= int_a or int_b;
47
48  int_ackA <= (int_a and Ack_out) or (int_ackA and (int_a or Ack_out));
49  AckA <= int_ackA;
50
51  int_ackB <= (int_b and Ack_out) or (int_ackB and( int_b or Ack_out));
52  AckB <= int_ackB;
53
54
55
56  end Behavioral;
```

```
1   --
        ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    21:25:52 05/21/2011
6   -- Design Name:
7   -- Module Name:    async_demux - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  --
        ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity async_demux is
31      Port ( Mux : in STD_LOGIC;
32                              Req_in : in  STD_LOGIC;
33             ReqA : out  STD_LOGIC;
34             ReqB : out  STD_LOGIC;
35             Ack_in : out  STD_LOGIC;
36             AckA : in  STD_LOGIC;
37             AckB : in  STD_LOGIC);
```

```
38 end async_demux;
39
40 architecture Behavioral of async_demux is
41 signal Int_A, Int_B : STD_LOGIC;
42 begin
43 Int_A<=  (Mux and Req_in);-- or (Int_A and (Mux or Req_in));
44 ReqA <= Int_A;
45
46 Int_B <= (not(Mux) and Req_in);-- or (Int_B and(Not(Mux) or Req_in));
47 ReqB <= Int_B;
48
49 Ack_in <= AckA or AckB;
50
51 end Behavioral;
```

C.2 VHDL for Synchronous Accu

C.2.1 Top module Accu.vhd

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company: DTU IMM
3  -- Engineer: Rasmus Madsen
4  --
5  -- Create Date:    10:59:11 09/08/2010
6  -- Design Name: Accu top component (structural)
7  -- Module Name:    accu - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 --
       ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity accu is
31     Port ( input : in  STD_LOGIC_VECTOR (7 downto 0);
32            clk : in STD_LOGIC;
33                          reset : in STD_LOGIC;
34                          outputs : out  STD_LOGIC_VECTOR (7 downto 0)
```

```
35                          );
36
37  end accu;
38
39  architecture Structural of accu is
40
41  component regist is
42      Port ( input : in  STD_LOGIC_VECTOR (7 downto 0);
43             output : out  STD_LOGIC_VECTOR (7 downto 0);
44             clk : in  STD_LOGIC;
45             reset : in  STD_LOGIC);
46  end component;
47
48  component  adder8bit is
49      Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
50             B : in  STD_LOGIC_VECTOR (7 downto 0);
51             Sum : out  STD_LOGIC_VECTOR (7 downto 0));
52  end component;
53
54  -- signals
55  signal Wire_sum, Wire_B,Wire_C : STD_LOGIC_VECTOR (7 downto 0);
56  begin
57
58
59  register1 : regist  port map(input =>input, output =>Wire_C, clk=>clk, reset
        =>reset);
60  register2 : regist  port map(input =>Wire_sum, output =>Wire_B, clk=>clk,
        reset=>reset);
61  adder1: adder8bit port map(A => Wire_C, B => Wire_B, Sum => Wire_sum);
62
63  outputs <= Wire_B;
64
65  end Structural  ;
```

C.2.2 Register.vhd

```
1   --
        ----------------------------------------------------------------------------------

2   -- Company: DTU IMM
3   -- Engineer: Rasmus Madse
4   --
5   -- Create Date:    11:21:56 09/06/2010
6   -- Design Name:  Rigister model
7   -- Module Name:    register - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  --
        ----------------------------------------------------------------------------------

20  library IEEE;
```

```
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity regist is
31     Port ( input : in  STD_LOGIC_VECTOR (7 downto 0);
32            output : out  STD_LOGIC_VECTOR (7 downto 0);
33            clk : in  STD_LOGIC;
34            reset : in  STD_LOGIC);
35 end regist;
36
37 architecture Behavioral of regist is
38
39 begin
40         process(clk, reset)
41                 begin
42                         if(reset = '1' ) then -- active high
43                                 output <= "00000000";
44                         elsif (clk ='1' and clk'EVENT) then
45                                 output <= input;
46                         end if;
47         end process;
48
49 end Behavioral;
```

C.2.3 Adder.vhd

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company: DTU IMM
3  -- Engineer: Rasmus Madsen
4  -- Create Date:    12:39:24 09/08/2010
5  -- Design Name: 8 bit register model
6  -- Module Name:    adder4bit - Behavioral
7  -- Project Name:
8  -- Target Devices:
9  -- Tool versions:
10 -- Description:
11 --
12 -- Dependencies:
13 --
14 -- Revision:
15 -- Revision 0.01 - File Created
16 -- Additional Comments:
17 --
18 --
       ----------------------------------------------------------------------------------
19 library IEEE;
20 use IEEE.STD_LOGIC_1164.ALL;
21 use IEEE.STD_LOGIC_ARITH.ALL;
22 use IEEE.STD_LOGIC_UNSIGNED.ALL;
23
24 ---- Uncomment the following library declaration if instantiating
```

```
25 ---- any Xilinx primitives in this code.
26 --library UNISIM;
27 --use UNISIM.VComponents.all;
28
29 entity adder8bit is
30     Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
31            B : in  STD_LOGIC_VECTOR (7 downto 0);
32            Sum : out  STD_LOGIC_VECTOR (7 downto 0));
33 end adder8bit;
34
35 architecture Behavioral of adder8bit is
36
37 begin
38
39 Sum <= A+B;
40
41 end Behavioral;
```

C.2.4 TestBench.vhd

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company:  DTU IMM
3  -- Engineer:Rasmus Madsen
4  --
5  -- Create Date:   12:33:58 01/19/2011
6  -- Design Name:   Test bench for Accu
7  -- Module Name:   C:/Rasmus/My Dropbox/3ugers/Accu_modeltest/Accutest.vhd
8  -- Project Name:  Accu_modeltest
9  -- Target Device:
10 -- Tool versions:
11 -- Description:
12 --
13 -- VHDL Test Bench Created by ISE for module: accu
14 --
15 -- Dependencies:
16 --
17 -- Revision:
18 -- Revision 0.01 - File Created
19 -- Additional Comments:
20 --
21 -- Notes:
22 -- This testbench has been automatically generated using types std_logic and
23 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
24 -- that these types always be used for the top-level I/O of a design in order
25 -- to guarantee that the testbench will bind correctly to the post-
       implementation
26 -- simulation model.
27 --
       ----------------------------------------------------------------------------------
28 LIBRARY ieee;
29 USE ieee.std_logic_1164.ALL;
30 USE ieee.std_logic_unsigned.all;
31 USE ieee.numeric_std.ALL;
32
33 ENTITY Accutest IS
34 END Accutest;
35
```

```
36  ARCHITECTURE behavior OF Accutest IS
37
38      -- Component Declaration for the Unit Under Test (UUT)
39
40      COMPONENT accu
41      PORT(
42           input : IN  std_logic_vector(7 downto 0);
43           clk : IN  std_logic;
44           reset : IN  std_logic;
45           outputs : OUT  std_logic_vector(7 downto 0)
46          );
47      END COMPONENT;
48
49
50     --Inputs
51     signal input : std_logic_vector(7 downto 0) := (others => '0');
52     signal clk : std_logic := '0';
53     signal reset : std_logic := '0';
54
55          --Outputs
56     signal outputs : std_logic_vector(7 downto 0);
57
58     -- Clock period definitions
59     constant clk_period : time := 10ns;
60
61  BEGIN
62
63          -- Instantiate the Unit Under Test (UUT)
64     uut: accu PORT MAP (
65            input => input,
66            clk => clk,
67            reset => reset,
68            outputs => outputs
69          );
70
71     -- Clock process definitions
72     clk_process :process
73     begin
74                  clk <= '0';
75                  wait for clk_period/2;
76                  clk <= '1';
77                  wait for clk_period/2;
78     end process;
79
80
81     -- Stimulus process
82     stim_proc: process
83     begin
84        -- hold reset state for 100ms.
85        wait for 10ns;
86
87        wait for clk_period*2;
88                  reset <='1';
89                  input <="00000000";
90
91                   wait for clk_period*2;
92                  reset <='0';
93                  input <="00000000";
94
95   wait for clk_period;
96                  reset <='0';
97                  input <="00000001";
98
99                   wait for clk_period;
100                 reset <='0';
```

```
101                 input <="00000011";
102
103                  wait for clk_period;
104                 reset <='0';
105                 input <="00000000";
106
107                  wait for clk_period;
108                 reset <='0';
109                 input <="00000010";
110
111
112
113       -- insert stimulus here
114
115       wait;
116    end process;
117
118 END;
```

C.3 VHLD for Asynchronous Accu

C.3.1 Top module Accu.vhd

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company: DTU IMM
3  -- Engineer: Rasmus Madsen
4  --
5  -- Create Date:    10:59:11 09/08/2010
6  -- Design Name: Accu top component (structural)
7  -- Module Name:    accu - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 --
       ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity accu is
```

```
31      Port ( input : in  STD_LOGIC_VECTOR (7 downto 0);
32             start : in STD_LOGIC;
33                           reset : in STD_LOGIC;
34                           outputs : out  STD_LOGIC_VECTOR (7 downto 0)
35                           );
36
37  end accu;
38
39  architecture Structural of accu is
40
41  component semi_decRingWinit is
42      Port (  Reset : in  STD_LOGIC;
43                           Req_in : in STD_LOGIC;
44                           Ack_in : out STD_LOGIC;
45                           Req_out : out STD_LOGIC;
46                           Ack_out : in STD_LOGIC;
47             Z : out  STD_LOGIC_VECTOR (1 downto 0));
48  end component semi_decRingWinit;
49
50  component Join is
51      Port ( Req_in : in  STD_LOGIC_VECTOR (1 downto 0);
52             Req_out : out  STD_LOGIC;
53             Ack_out : in  STD_LOGIC;
54             Ack_in : out  STD_LOGIC_VECTOR (1 downto 0));
55  end component Join;
56
57  component Fork is
58      Port ( Req_in : in  STD_LOGIC;
59             Req_out : out  STD_LOGIC_VECTOR (1 downto 0);
60             Ack_out : in  STD_LOGIC_VECTOR (1 downto 0);
61             Ack_in : out  STD_LOGIC);
62  end component Fork;
63
64  component latch is
65      Port ( Reset : in  STD_LOGIC;
66                           CTRL  : in  STD_LOGIC;
67             input : in  STD_LOGIC_VECTOR (7 downto 0);
68             output : out  STD_LOGIC_VECTOR (7 downto 0));
69  end component;
70
71
72
73  component  adder8bit is
74      Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
75             B : in  STD_LOGIC_VECTOR (7 downto 0);
76             Sum : out  STD_LOGIC_VECTOR (7 downto 0));
77  end component;
78
79  -- signals
80  signal Wire_sum, Wire_A, Wire_B,Wire_C, Btw_latch1, Btw_latch2 :
        STD_LOGIC_VECTOR (7 downto 0);
81  signal x_req,x_ack,y_req,y_ack, z_req,z_ack : std_logic;
82  signal join_req_in, join_ack_in,fork_req_out, fork_ack_out : std_logic_vector
        (1 downto 0);
83  signal join_req_out, join_ack_out, fork_req_in, fork_ack_in: std_logic;
84  signal M1, S1, M2, S2: std_logic;
85  signal delay_in, delay_out : std_logic;
86
87  begin
88  Wire_A <= input;
89
90  delay_out <= delay_in after 8ns;
91
92  x_req <= not(x_ack) after 2 ns; --and start after 2ns;
93  z_ack <= z_req after 4ns;
```

```
94
95  Latch_CTRL1 : semi_decRingWinit port map( Reset=>reset,
96                                                                  Req_in
                                                                         =>
                                                                         x_req
                                                                         ,
97                                                                Ack_in
                                                                       =>
                                                                       x_ack
                                                                       ,
98                                                                Req_out
                                                                       =>
                                                                       join_req_in
                                                                       (0)
                                                                       ,
99                                                                Ack_out
                                                                       =>
                                                                       join_ack_in
                                                                       (0)
                                                                       ,
100                                                               Z
                                                                       (0)
                                                                       =>
                                                                       M1
                                                                       ,
101                                                                Z
                                                                       (1)
                                                                       =>
                                                                       S1
                                                                       )
                                                                       ;
102
103 Join1 : Join port map (Req_in=>join_req_in,
104                                                  Req_out=>delay_in,
105                                                  Ack_out=>y_ack,
106                                                  Ack_in=>join_ack_in
                                                       );
107
108 Latch_CTRL2 : semi_decRingWinit port map( Reset=>reset,
109                                                                 Req_in
                                                                       =>
                                                                       delay_out
                                                                       ,
110                                                               Ack_in
                                                                       =>
                                                                       y_ack
                                                                       ,
111                                                               Req_out
                                                                       =>
                                                                       fork_req_in
                                                                       ,
```

```
Ack_out => fork_ack_ ,
Z (0) => M2 ,
Z (1) => S2 ) ;
Fork1 : Fork port map( Req_in=> fork_req_in ,
                       Req_out (0)=> z_req ,
                       Req_out (1)=> join_req_in (1) ,
                       Ack_out (0)=> z_ack ,
                       Ack_out (1)=> join_ack_in (1) ,
                       Ack_in=>fork_ack_in );

Master1 : latch  port map(input =>Wire_A , output =>Btw_latch1 , CTRL=>M1 ,
     Reset=>reset);
Slave1 : latch port map(input =>Btw_latch1  , output=>Wire_C , CTRL  =>S1 , Reset
      => reset);
Master2 : latch port map( input =>Wire_sum , output =>Btw_latch2 , CTRL=>M2 ,
     Reset  =>reset);
Slave2 : latch  port map(input => Btw_latch2 , output  =>Wire_B , CTRL=>S2 ,
     Reset=>reset);
adder1: adder8bit port map(A => Wire_C , B => Wire_B , Sum => Wire_sum);

outputs <= Wire_B;

end Structural  ;
```

C.3.2 latchcontrol top module.vhd

C.3.3 Latch control single latch.vhd

```
--
     ------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:05:49 11/12/2010
-- Design Name:
-- Module Name:    SemiDecoupledV2 - Behavioral
-- Project Name:
```

```
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SemiDecoupledV3 is
    Port ( Reset : in  STD_LOGIC;
                          Init : in STD_LOGIC;
           Req_in : in  STD_LOGIC;
           Ack_in : in  STD_LOGIC;
           Req_out : out STD_LOGIC;
           Ack_out : out  STD_LOGIC;
           Z : out  STD_LOGIC);
end SemiDecoupledV3;

architecture Behavioral of SemiDecoupledV3 is

Signal A, Rout_int, Z_int : STD_LOGIC;
begin

--PROCESS is
Semi:  A <= Init when Reset = '1' else
                        '1' after 2ns when (Req_in and not Rout_int)= '1'
                            else
                        '0' after 1ns when (not Req_in and Rout_int and
                            Ack_in) = '1';

                Rout_int <= Init when Reset = '1' else
                                    '1' after 2ns when (A and not Ack_in
                                        ) = '1' else
                                    '0' after 1ns when A = '0' ;


                Z_int <= Init when Reset ='1' else
                                    '1' after 2ns when A = '1' else
                                    '0' after 1ns when A = '0';

                Ack_out <= '0' when Reset = '1' else
                                      '1' after 2ns when Z_int = '1'
                                          else
                                  '0' after 1ns when Z_int = '0';
               Req_out <= Rout_int;
               Z <= Z_int;
--end process

end Behavioral;
```

C.3.4 Adder

C.3.5 TestBench.vhd

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:   15:02:30 03/17/2011
6  -- Design Name:
7  -- Module Name:   C:/Rasmus/My Dropbox/Speciale/Projects/Accu/Vhdl_async/
       Async_accu/Test_async_accu.vhd
8  -- Project Name:  Async_accu
9  -- Target Device:
10 -- Tool versions:
11 -- Description:
12 --
13 -- VHDL Test Bench Created by ISE for module: accu
14 --
15 -- Dependencies:
16 --
17 -- Revision:
18 -- Revision 0.01 - File Created
19 -- Additional Comments:
20 --
21 -- Notes:
22 -- This testbench has been automatically generated using types std_logic and
23 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
24 -- that these types always be used for the top-level I/O of a design in order
25 -- to guarantee that the testbench will bind correctly to the post-
       implementation
26 -- simulation model.
27 --
       ----------------------------------------------------------------------------------
28 LIBRARY ieee;
29 USE ieee.std_logic_1164.ALL;
30 USE ieee.std_logic_unsigned.all;
31 USE ieee.numeric_std.ALL;
32
33 ENTITY Test_async_accu IS
34 END Test_async_accu;
35
36 ARCHITECTURE behavior OF Test_async_accu IS
37
38     -- Component Declaration for the Unit Under Test (UUT)
39
40     COMPONENT accu
41     PORT(
42          input : IN  std_logic_vector(7 downto 0);
43          reset : IN  std_logic;
44          outputs : OUT  std_logic_vector(7 downto 0)
45         );
46     END COMPONENT;
47
48
49    --Inputs
50    signal input : std_logic_vector(7 downto 0) := (others => '0');
51    signal reset : std_logic := '0';
52
```

```
53          --Outputs
54      signal outputs : std_logic_vector(7 downto 0);
55
56  BEGIN
57
58          -- Instantiate the Unit Under Test (UUT)
59      uut: accu PORT MAP (
60             input => input,
61             reset => reset,
62             outputs => outputs
63           );
64
65      -- No clocks detected in port list. Replace <clock> below with
66      -- appropriate port name
67
68
69      -- Stimulus process
70      stim_proc: process
71      begin
72         -- hold reset state for 100ms.
73         wait for 10ns;
74                    input<="00000000";
75                    reset<='0';
76
77                    wait for 140ns;
78                    input<="00000000";
79                    reset<='1';
80
81                    wait for 100ns;
82                    input<="00000000";
83                    reset<='0';
84
85                    wait for 100ns;
86                    input<="00000001";
87                    reset<='0';
88
89                    wait for 200ns;
90                    input<="00000101";
91                    reset<='0';
92
93
94
95         wait;
96      end process;
97
98  END;
```

C.4 VHDL code for Synchronous GCD

C.5 VHDL code for asynchronous GCD simple desynchronization

```
1  --
       --------------------------------------------------------------------------------
```

```
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    12:31:06 12/15/2010
6  -- Design Name:
7  -- Module Name:    GCD_async - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 --
     ----------------------------------------------------------------------------------

20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity GCD_async is
31     Port ( Reset: in STD_LOGIC;
32                           input_valid : in  STD_LOGIC;
33            Data_in : in  STD_LOGIC_VECTOR (7 downto 0);
34            Req_in : in  STD_LOGIC;
35            Req_out : out  STD_LOGIC;
36            Ack_in : in  STD_LOGIC;
37            Ack_out : out  STD_LOGIC;
38            output_valid : out  STD_LOGIC;
39            Data_out : out  STD_LOGIC_VECTOR (7 downto 0));
40 end GCD_async;
41
42 architecture Structural of GCD_async is
43
44 Component  Latch is
45 Port ( Reset : in  STD_LOGIC;
46                           CTRL  : in  STD_LOGIC;
47            input : in  STD_LOGIC_VECTOR (7 downto 0);
48            output : out  STD_LOGIC_VECTOR (7 downto 0));
49 end component;
50
51 component Latch3bit is
52 Port ( Reset : in  STD_LOGIC;
53                           CTRL  : in  STD_LOGIC;
54            input : in  STD_LOGIC_VECTOR (2 downto 0);
55            output : out  STD_LOGIC_VECTOR (2 downto 0));
56 end component;
57
58 -- simple controller
59 component semi_decRingWinit is
60     Port ( Reset : in  STD_LOGIC;
61                           Req_in : in STD_LOGIC;
62                           Ack_in : out STD_LOGIC;
63                           Req_out : out STD_LOGIC;
64                           Ack_out : in STD_LOGIC;
```

```
65            Z : out  STD_LOGIC_VECTOR (1 downto 0));
66  end component;
67
68  --- implemented controller --
69  component Latch_control_top is
70      Port ( Reset : in  STD_LOGIC;
71             Req_in : in  STD_LOGIC;
72             Ack_in : out  STD_LOGIC;
73             Req_out : out  STD_LOGIC;
74             Ack_out : in  STD_LOGIC;
75             Z : out  STD_LOGIC_VECTOR (1 downto 0));
76  end component Latch_control_top;
77
78  component gcd_core IS
79      PORT (--clk:      IN std_logic;                          -- The clock
            signal.
80            reset:    IN std_logic;                          -- Reset the module
                .
81            req:      IN std_logic;                          -- Start
                computation. (DATA IN)
82            AB:       IN std_logic_vector(7 downto 0);       -- The two
                operands. ( DATA IN)
83            ack:      OUT std_logic;                         -- Computation is
                complete.
84            C:        OUT std_logic_vector(7 downto 0); -- The result.
85                      reg_a:     IN std_logic_vector(7 downto 0);
86                      next_reg_a: OUT std_logic_vector(7 downto 0);
87                      reg_b:     IN std_logic_vector(7 downto 0);
88                      next_reg_b: OUT std_logic_vector(7 downto 0);
89                      state:     IN std_logic_vector(2 downto 0);
90                      next_state: OUT std_logic_vector(2 downto 0));
91  END component;
92
93  component Fork is
94      Port ( Req_in : in  STD_LOGIC;
95             Req_out : out  STD_LOGIC_VECTOR (1 downto 0);
96             Ack_in : in  STD_LOGIC_VECTOR (1 downto 0);
97             Ack_out : out  STD_LOGIC);
98  end component;
99
100 component Join is
101     Port ( Req_in : in  STD_LOGIC_VECTOR (1 downto 0);
102            Req_out : out  STD_LOGIC;
103            Ack_in : in  STD_LOGIC;
104            Ack_out : out  STD_LOGIC_VECTOR (1 downto 0));
105 end component;
106 component DelayElement is
107     Port ( xin : in  STD_LOGIC;
108            delayed_x : out  STD_LOGIC);
109 end component DelayElement;
110
111 signal Int_regA, Int_NextA,Int_regB, Int_NextB : std_logic_vector(7 downto 0)
        ;
112 signal IntState, Int_NextState : std_logic_vector(2 downto 0);
113 signal Latch_M, Latch_S : STD_LOGIC;
114 signal L_Int_A, L_Int_B : STD_LOGIC_vector(7 downto 0);
115 signal L_Int_State: STD_LOGIC_VECTOR(2 downto 0);
116 signal Int_req_in, Int_req_out, int_ack,req_Join2control, req_control2fork :
        std_logic;
117 signal ack_control2join, ack_fork2control : STD_LOGIC;
118 begin
119 --Int_req_in <= Int_req_out after 2ns; -- Delay element
120 Delay : DelayElement port map(xin=> Int_req_out,
121                                                                  delayed_x
```

```
                                                                    =>
                                                                    Int_req_in
                                                                    )
                                                                    ;

CORE: gcd_core port map (reset => Reset, req => input_valid, AB => Data_in,
    ack => output_valid, C => Data_out,
                                                  reg_a => Int_regA
                                                       , next_reg_a
                                                        =>
                                                       Int_NextA,
                                                       reg_b =>
                                                       Int_regB,
                                                       next_reg_b
                                                       => Int_NextB
                                                       ,
                                                        state =>
                                                             IntState
                                                             ,
                                                             next_state
                                                              =>
                                                             Int_NextState
                                                             );

join1 : Join port map( Req_in(0) => Req_in,
                                                 Req_in(1) =>
                                                      Int_req_in,
                                                 Req_out =>
                                                      req_Join2control
                                                      ,
                                                 Ack_in =>
                                                      ack_control2join
                                                      ,
                       Ack_out(0) => Ack_out,
                                                 Ack_out(1) =>
                                                      Int_ack);

fork1 : fork port map (Req_in => req_control2fork,
                                                  Req_out(0) =>
                                                       Req_out,
                                                  Req_out(1) =>
                                                       Int_req_out,
                                                  Ack_in(0) => Ack_in
                                                       ,
                                                  Ack_in(1) =>
                                                       Int_ack,
                                                  Ack_out =>
                                                       ack_fork2control
                                                       );

Latc_control : Latch_control_top port map (Reset => Reset,  --
    semi_decRingWinitLatch_control_top
                                                                                                  Req_in

                                                                                                       =>

                                                                                                       req
                                                                                                       ,
                                                                                                  Ack_out

                                                                                                       =>

                                                                                                       ack
```

```
,
Req_out => req_cont,
Ack_in => ack_cont,
Z(0) => Latch_M,
Z(1) => Latch_S);

L_A_M : Latch port map (Reset => Reset,
                        CTRL  => Latch_M,
                        input => Int_NextA,
                        output => L_Int_A);

L_A_S : Latch port map (Reset => Reset,
                        CTRL  => Latch_S,
                        input => L_Int_A,
                        output => Int_RegA);

L_B_M : Latch port map (Reset => Reset,
                        CTRL  => Latch_M,
                        input => Int_NextB,
                        output => L_Int_B);

L_B_S : Latch port map (Reset => Reset,
```

```
165                                                          CTRL  =>
                                                                 Latch_S,
166                                                          input =>
                                                                 L_Int_B,
167                                                          output =>
                                                                 Int_RegB
                                                                 );
168
169 L_S_M : Latch3bit port map ( Reset => Reset,
170                                                                 CTRL
                                                                       =>
                                                                       Latch_M
                                                                       ,
171                                                                 input
                                                                       =>
                                                                       Int_NextState
                                                                       ,
172                                                                 output
                                                                       =>
                                                                       L_Int_State
                                                                       )
                                                                       ;
173
174 L_S_S : Latch3bit port map ( Reset => Reset,
175                                                                 CTRL
                                                                       =>
                                                                       Latch_S
                                                                       ,
176                                                                 input
                                                                       =>
                                                                       L_Int_State
                                                                       ,
177                                                                 output
                                                                       =>
                                                                       IntState
                                                                       )
                                                                       ;
178
179
180
181 end Structural;
```

C.6 VHDL code for asynchronous GCD simple desynchronization

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    12:31:06 12/15/2010
6  -- Design Name:
7  -- Module Name:    GCD_async - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 --
       ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity GCD_async is
31     Port ( Reset: in STD_LOGIC;
32                           input_valid : in  STD_LOGIC;
33            Data_in : in  STD_LOGIC_VECTOR (7 downto 0);
34            Req_in : in  STD_LOGIC;
35            Req_out : out  STD_LOGIC;
36            Ack_in : in  STD_LOGIC;
37            Ack_out : out  STD_LOGIC;
38            output_valid : out  STD_LOGIC;
39            Data_out : out  STD_LOGIC_VECTOR (7 downto 0));
40 end GCD_async;
41
42 architecture Structural of GCD_async is
43
44 Component  Latch is
45 Port ( Reset : in  STD_LOGIC;
46                           CTRL  : in  STD_LOGIC;
47            input : in  STD_LOGIC_VECTOR (7 downto 0);
48            output : out  STD_LOGIC_VECTOR (7 downto 0));
49 end component;
50
51 component Latch3bit is
52 Port ( Reset : in  STD_LOGIC;
53                           CTRL  : in  STD_LOGIC;
54            input : in  STD_LOGIC_VECTOR (2 downto 0);
55            output : out  STD_LOGIC_VECTOR (2 downto 0));
56 end component;
57
58 -- simple controller
59 component semi_decRingWinit is
60     Port ( Reset : in  STD_LOGIC;
```

```
61                          Req_in : in STD_LOGIC;
62                          Ack_in : out STD_LOGIC;
63                          Req_out : out STD_LOGIC;
64                          Ack_out : in STD_LOGIC;
65            Z : out  STD_LOGIC_VECTOR (1 downto 0));
66  end component;
67
68  --- implemented controller --
69  component Latch_control_top is
70      Port ( Reset : in  STD_LOGIC;
71             Req_in : in  STD_LOGIC;
72             Ack_in : out  STD_LOGIC;
73             Req_out : out  STD_LOGIC;
74             Ack_out : in  STD_LOGIC;
75             Z : out  STD_LOGIC_VECTOR (1 downto 0));
76  end component Latch_control_top;
77
78  component gcd_core IS
79      PORT (--clk:      IN std_logic;                          -- The clock
            signal.
80            reset:    IN std_logic;                          -- Reset the module
                .
81            req:      IN std_logic;                          -- Start
                computation. (DATA IN)
82            AB:       IN std_logic_vector(7 downto 0);       -- The two
                operands. ( DATA IN)
83            ack:      OUT std_logic;                         -- Computation is
                complete.
84            C:        OUT std_logic_vector(7 downto 0); -- The result.
85                        reg_a:     IN std_logic_vector(7 downto 0);
86                        next_reg_a: OUT std_logic_vector(7 downto 0);
87                        reg_b:     IN std_logic_vector(7 downto 0);
88                        next_reg_b: OUT std_logic_vector(7 downto 0);
89                        state:     IN std_logic_vector(2 downto 0);
90                        next_state: OUT std_logic_vector(2 downto 0));
91  END component;
92
93  component Fork is
94      Port ( Req_in : in  STD_LOGIC;
95             Req_out : out  STD_LOGIC_VECTOR (1 downto 0);
96             Ack_in : in  STD_LOGIC_VECTOR (1 downto 0);
97             Ack_out : out  STD_LOGIC);
98  end component;
99
100 component Join is
101     Port ( Req_in : in  STD_LOGIC_VECTOR (1 downto 0);
102            Req_out : out  STD_LOGIC;
103            Ack_in : in  STD_LOGIC;
104            Ack_out : out  STD_LOGIC_VECTOR (1 downto 0));
105 end component;
106
107 signal Int_regA, Int_NextA,Int_regB, Int_NextB : std_logic_vector(7 downto 0)
        ;
108 signal IntState, Int_NextState : std_logic_vector(2 downto 0);
109 signal Latch_M, Latch_S : STD_LOGIC;
110 signal L_Int_A, L_Int_B : STD_LOGIC_vector(7 downto 0);
111 signal L_Int_State: STD_LOGIC_VECTOR(2 downto 0);
112 signal Int_req_in, Int_req_out, int_ack,req_Join2control, req_control2fork :
        std_logic;
113 signal ack_control2join, ack_fork2control : STD_LOGIC;
114 begin
115 Int_req_in <= Int_req_out after 5ns; -- Delay element
116
117 CORE: gcd_core port map (reset => Reset, req => input_valid, AB => Data_in,
        ack => output_valid, C => Data_out,
```

```
118                                                   reg_a => Int_regA
                                                           , next_reg_a
                                                            =>
                                                           Int_NextA,
                                                           reg_b =>
                                                           Int_regB,
                                                           next_reg_b
                                                           => Int_NextB
                                                           ,
119                                                         state =>
                                                                 IntState
                                                                 ,
                                                                 next_state
                                                                  =>
                                                                 Int_NextState
                                                                 );
120
121 join1 : Join port map( Req_in(0) => Req_in,
122                                                 Req_in(1) =>
                                                        Int_req_in,
123                                                 Req_out =>
                                                        req_Join2control
                                                        ,
124                                                 Ack_in =>
                                                        ack_control2join
                                                        ,
125                       Ack_out(0) => Ack_out,
126                                                 Ack_out(1) =>
                                                        Int_ack);
127
128 fork1 : fork port map (Req_in => req_control2fork,
129                                                  Req_out(0) =>
                                                         Req_out,
130                                                  Req_out(1) =>
                                                         Int_req_out,
131                                                  Ack_in(0) => Ack_in
                                                         ,
132                                                  Ack_in(1) =>
                                                         Int_ack,
133                                                  Ack_out =>
                                                         ack_fork2control
                                                         );
134
135 Latc_control : semi_decRingWinit port map (Reset => Reset,  --
        semi_decRingWinitLatch_control_top
136                                                                                                  Req_in
                                                                                                       =>
                                                                                                       req_Join
                                                                                                       ,
137                                                                                                  Ack_out
                                                                                                       =>
                                                                                                       ack_fork
                                                                                                       ,
138                                                                                                  Req_out
                                                                                                       =>
                                                                                                       req_cont
                                                                                                       ,
```

```
                                        Ack_in => ack
                                        ,
                                        Z (0) => Lat
                                        ,
                                        Z (1) => Lat
                                        );

L_A_M : Latch port map (Reset => Reset,
                        CTRL  => Latch_M,
                        input => Int_NextA,
                        output => L_Int_A);

L_A_S : Latch port map (Reset => Reset,
                        CTRL  => Latch_S,
                        input => L_Int_A,
                        output => Int_RegA);

L_B_M : Latch port map (Reset => Reset,
                        CTRL  => Latch_M,
                        input => Int_NextB,
                        output => L_Int_B);

L_B_S : Latch port map (Reset => Reset,
                        CTRL  => Latch_S,
                        input => L_Int_B,
                        output => Int_RegB);

L_S_M : Latch3bit port map ( Reset => Reset,
```

```
164                                                                 CTRL
                                                                         =>
                                                                    Latch_M
                                                                    ,
165                                                                 input
                                                                         =>
                                                                    Int_NextState
                                                                    ,
166                                                                 output
                                                                         =>
                                                                    L_Int_State
                                                                    )
                                                                    ;
167
168 L_S_S : Latch3bit port map ( Reset => Reset,
169                                                                 CTRL
                                                                         =>
                                                                    Latch_S
                                                                    ,
170                                                                 input
                                                                         =>
                                                                    L_Int_State
                                                                    ,
171                                                                 output
                                                                         =>
                                                                    IntState
                                                                    )
                                                                    ;
172
173
174
175 end Structural;
```

Appendix D

VHDL Edge detection

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company: course 02154
3  -- Engineer: Rasmus Madsen s033176
4  --
5  -- Create Date:    13:20:28 11/16/2008
6  -- Design Name:
7  -- Module Name:    BitFlipper - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -- 4 pixel flipper. this unit can be used to reverse the orde of the 4 inputs
       . sot that ABCD -> DCBA
20 --
       ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use IEEE.STD_LOGIC_ARITH.ALL;
24 use IEEE.STD_LOGIC_UNSIGNED.ALL;
25
26 ---- Uncomment the following library declaration if instantiating
27 ---- any Xilinx primitives in this code.
28 --library UNISIM;
```

```
29 --use UNISIM.VComponents.all;
30
31 entity BitFlipper is
32     Port ( Word_in : in  STD_LOGIC_VECTOR (31 downto 0);
33            flip : in  STD_LOGIC;
34            Word_out : out  STD_LOGIC_VECTOR (31 downto 0));
35 end BitFlipper;
36
37 architecture Behavioral of BitFlipper is
38
39 begin
40
41 flipbits : process(flip,Word_in) is
42 begin
43         case flip is
44                 when '1' =>
45                         Word_out(31 downto 24) <= Word_in(7 downto 0);
46                         Word_out(23 downto 16) <= Word_in(15 downto 8);
47                         Word_out(15 downto 8)  <= Word_in(23 downto 16);
48                         Word_out(7 downto 0)   <= Word_in(31 downto 24);
49                 when others =>
50                         Word_out(31 downto 24) <= Word_in(31 downto 24);
51                         Word_out(23 downto 16) <= Word_in(23 downto 16);
52                         Word_out(15 downto 8)  <= Word_in(15 downto 8);
53                         Word_out(7 downto 0)   <= Word_in(7 downto 0);
54                 end case;
55 end process flipbits;
56 end Behavioral;
```

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    14:19:30 05/09/2011
6  -- Design Name:
7  -- Module Name:    count4bit_FF - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 --
       ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity count4bit_FF is
31     Port ( clk : in  STD_LOGIC;
```

```
32            lastval_in : in  STD_LOGIC_VECTOR (3 downto 0);
33            count_in : in  STD_LOGIC_VECTOR (3 downto 0);
34            lastval_out : out  STD_LOGIC_VECTOR (3 downto 0);
35            count_out : out  STD_LOGIC_VECTOR (3 downto 0);
36            targetR_in : in  STD_LOGIC;
37            targetR_out : out  STD_LOGIC);
38 end count4bit_FF;
39
40 architecture Behavioral of count4bit_FF is
41
42 begin
43 clocked : Process(clk, lastval_in, count_in, targetR_in)
44         begin
45                 if (clk'event and clk ='1') then
46                 count_out <= count_in;
47                 lastval_out <= lastval_in;
48                 targetR_out <= targetR_in;
49                 end if;
50         end process;
51 end Behavioral;
```

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company: course 02154
3  -- Engineer: Rasmus madsen
4  --
5  -- Create Date:    22:30:48 11/16/2008
6  -- Design Name:
7  -- Module Name:    counter_4bit - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description: 4bit counter with syncronous reset/load
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 -- should be made into one generic counter when design works
19 --
       ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity counter2bit is
31     Port ( --clk : in  STD_LOGIC;
32            reset : in  STD_LOGIC; -- active high
33                        pause : in STD_LOGIC;
34                        countLow_in : in STD_LOGIC_VECTOR (1 downto 0);
35                        countHigh_in : in STD_LOGIC_VECTOR (6 downto 0);
36            count_out_low : out  STD_LOGIC_VECTOR (1 downto 0);
37                        count_out_high :out STD_LOGIC_VECTOR( 6 downto 0)
38                        );
39 end counter2bit;
```

```
40
41  architecture Behavioral of counter2bit is
42
43  --signal tempcountLow : STD_LOGIC_VECTOR (1 downto 0);
44  --signal tempcountHigh : integer := 0;
45  --signal tempcountHigh : STD_LOGIC_VECTOR( 6 downto 0);
46
47  begin
48   count : process(reset,pause,countLow_in,countHigh_in)
49          begin
50                          --if clk = '1' and clk'event then
51                                  if reset ='1' then
52                                          count_out_low <= "00";
53                                          count_out_high<= "0000000" ;--0 ;--
                                                tempcountHigh;
54                                  elsif pause ='1' then
55                                          count_out_low <= countLow_in ;
56                                          count_out_high<= countHigh_in;
57                                  elsif countLow_in ="10" then
58                                          count_out_low <= "00";
59                                          if (countHigh_in = "1011001")then--
                                                89 ) then --90 column= 0-89
60                                                  count_out_high <= "0000000";
                                                        --0
61                                          else
62                                          count_out_high<=countHigh_in+1;
63                                          end if;
64                                  else
65                                          count_out_low <= countLow_in+1;
66                                                  count_out_high<=countHigh_in;
67                                  end if;
68                          --end if;
69
70          end process;
71          --count_out_low <= tempcountLOW;
72          --count_out_high <= conv_std_logic_vector(tempcountHigh,7);
73          --count_out_high <=tempcountHigh;
74
75
76  end Behavioral;
```

```
1   --
        ----------------------------------------------------------------------------------
2   -- Company: course 02154
3   -- Engineer: Rasmus madsen & Morten R. Jørgensen
4   --
5   -- Create Date:    22:15:40 11/17/2008
6   -- Design Name:
7   -- Module Name:    EdgeTop - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  --
        ----------------------------------------------------------------------------------
```

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity FSMTop is
31     Port ( --clk : in  STD_LOGIC; -- not used anymore
32                         reset : in std_logic;
33                         start : in  STD_LOGIC;
34                         hold : in STD_LOGIC;
35                         done : out  STD_LOGIC; --x
36
37
38                         Regcount : out STD_LOGIC_VECTOR (3 downto 0); --x
39
40                         -- FIFO control --
41                         FIFOadress :out STD_logic_vector (1 downto 0); --x
42                         FIFOread : out STD_LOGIC; --x
43                         FIFOwrite : out STD_LOGIC; --x
44                         -- mem counters --
45                         MemColumCountOut: out STD_LOGIC_vector(6 downto 0);
                               --x
46                         OffsetOut : out STD_LOGIC_VECTOR (1 downto 0); --x
47                         Rowcount : out STD_LOGIC_VECTOR (8 downto 0); --x
48                         -- write counter --
49                         WriteAddress :  out std_logic_vector (31 downto 0);
                               --x
50                         read_write: out std_logic; -- 0 read, 1 = write --x
51                         --test--
52                         PxNr_out: out STD_LOGIC_VECTOR(8 downto 0); --x
53                         stateout : out STD_LOGIC_VECTOR(3 downto 0);--
                               testing purpose only x
54
55                         write_req : out std_logic;
56                         write_ack : in std_logic;
57                         offset_req : out std_logic;
58                         offset_ack : in std_logic;
59                         px_req : out std_logic;
60                         px_ack : in std_logic;
61                         reg_req : out std_logic;
62                         reg_ack : in std_logic;
63                         fsm_req : out STD_LOGIC;
64                         fsm_ack :in STD_LOGIC
65                         );
66 end FSMTop;
67
68 architecture Behavioral of FSMTop is
69 ---------------signals here ---------------
70 signal regSel,regPs,regVal : STD_LOGIC;
71 signal bufSel,bufPs,bufVal : STD_LOGIC;
72 signal columnSel,columnPs,columnVal : STD_LOGIC;
73 signal rowPs,rowSel,rowVal : STD_LOGIC;
74 signal regL,bufL : STD_LOGIC_VECTOR(3 downto 0);
75 signal columnL : STD_LOGIC_VECTOR(8 downto 0);
76 signal rowL : STD_LOGIC_VECTOR(8 downto 0);
77 --signal MemRCount : STD_LOGIC_VECTOR(8 downto 0);
78 --signal MemCCount :STD_LOGIC_VECTOR(6 downto 0);
79 signal ResetOffsetCounter :STD_LOGIC;
80 signal ResetMemCCount :STD_LOGIC;
81 signal OffsetPause : STD_LOGIC;
```

```
82  signal MemColPause : STD_LOGIC;
83  signal Wreset : STD_LOGIC;
84  Signal Wpause : STD_LOGIC;
85  signal int_state, int_sxt_state, int_mux_in, int_stateout : STD_LOGIC_VECTOR
        (3 downto 0);
86  signal buf_lastval_in, buf_count_in, buf_lastval_out, buf_count_out :
        STD_LOGIC_VECTOR( 3 downto 0);
87  signal buf_target_in : STD_LOGIC;
88  signal reg_lastval_in, reg_count_in, reg_lastval_out, reg_count_out :
        STD_LOGIC_VECTOR( 3 downto 0);
89  signal reg_target_in : STD_LOGIC;
90  signal temp_reached_row_in,temp_reached_column_in, temp_reached_row_out,
        temp_reached_column_out  :STD_LOGIC;
91  signal temp_count_row_in, temp_count_column_in, temp_count_row_out,
        temp_count_column_out :STD_LOGIC_VECTOR(8 downto 0);
92  signal temp_count_low_in, temp_count_low_out  : STD_LOGIC_VECTOR( 1 downto 0)
        ;
93  signal temp_count_high_in, temp_count_high_out: STD_LOGIC_VECTOR( 6 downto 0)
        ;
94  signal Wcount_in, Wcount_mem_in, Wcount_out : STD_LOGIC_VECTOR(31 downto 0);
95  signal NotReset : STD_LOGIC;
96  ------request and acknowledge signals
        ---------------------------------------------
97  signal Write_req_in, Write_ack_in, write_req_out, write_ack_out : std_logic;
        -- Write
98  signal offset_req_in, offset_ack_in, offset_req_out, offset_ack_out :
        std_logic; -- offset
99  signal px_req_in, px_ack_in, px_req_out, px_ack_out: std_logic; -- pxcounter
100 signal reg_req_in, reg_ack_in, reg_req_out, reg_ack_out: std_logic; -- re
        counter
101 signal buf_req_in, buf_ack_in, buf_req_out, buf_ack_out : std_logic; -- buf
102 signal fsm_req_del, fsm_ack_in, fsm_req_out, fsm_ack_out: std_logic;
103
104 signal Fork_write_req_out, Fork_write_ack_out : STD_LOGIC_VECTOR(1 downto 0);
105 signal Fork_req_delay: STD_LOGIC;
106 signal offset_write_req_out, offset_write_ack_out : STD_LOGIC_VECTOR(1 downto
        0);
107 signal offset_req_delay: STD_LOGIC;
108 signal px_write_req_out, px_write_ack_out : STD_LOGIC_VECTOR(1 downto 0);
109 signal px_req_delay: STD_LOGIC;
110 signal reg_write_req_out, reg_write_ack_out : STD_LOGIC_VECTOR(1 downto 0);
111 signal reg_req_delay: STD_LOGIC;
112 signal buf_req_delay: STD_LOGIC;
113
114 signal fsm_7Req, fsm_7Ack : std_logic_vector(6 downto 0);
115
116 signal CTRL_upcount32bit, CTRL_counter2bit, CTRL_counter9bit, CTRL_buf,
        CTRL_reg, CTRL_FSM : STD_LOGIC_VECTOR(1 downto 0);
117 ---------------components here-------------------
118 component FSM_latch is
119           port (CTRL : in STD_LOGIC_VECTOR(1 downto 0); -- only difference (
                clk)
120                         hold : in STD_LOGIC;
121            reset : in  STD_LOGIC;
122            Next_state : in  STD_LOGIC_VECTOR (3 downto 0);
123            state : out  STD_LOGIC_VECTOR (3 downto 0);
124                           muxCtrl: in  STD_LOGIC_VECTOR (3 downto 0);
125                           stateout: out  STD_LOGIC_VECTOR (3 downto 0)  );
126 end component FSM_latch;
127
128 component fork7port is
129     Port ( Req_in : in  STD_LOGIC;
130            Ack_in : out  STD_LOGIC;
131            Req_out : out  STD_LOGIC_VECTOR (6 downto 0);
132            Ack_out : in  STD_LOGIC_VECTOR (6 downto 0));
```

```
133 end component fork7port;
134
135 component  muxCTRL is
136     Port ( next_state : in  STD_LOGIC_VECTOR (3 downto 0);
137            muxCtrl_in : in  STD_LOGIC_VECTOR (3 downto 0);
138            muxCtrl : out  STD_LOGIC_VECTOR (3 downto 0));
139 end component muxCTRL;
140
141 component FSM is
142    Port ( --clk : in  STD_LOGIC;
143                         --reset: in STD_LOGIC;
144                         start: in std_logic;
145                        -- hold : in STD_LOGIC;
146                         -- FIFO control signals --
147                         FIFOadress :out STD_logic_vector (1 downto 0);
148                         FIFOread : out STD_LOGIC;
149                         FIFOwrite : out STD_LOGIC;
150                         -- register counter 0-8 + 15( 15 = no register)--
151
152                         regSelectReset        : out STD_LOGIC;
153                         regPause                    :  out std_logic;
154                         regLoad                     : out
                                STD_LOGIC_VECTOR (3 downto 0);
155                       --  regValReached       : in STD_LOGIC;
156
157                         --- px column counter 0-351 --
158                         columnReset                 : out STD_LOGIC;
159                         columnPause                 : out STD_LOGIC;
160                         columnLoad                  : out
                                STD_LOGIC_VECTOR (8 downto 0);
161                         columnValueReached : in STD_LOGIC;
162                         --px row counter 0-287 --
163                         rowReset                    : out STD_LOGIC;
164                         rowPause                    : out STD_LOGIC;
165                         rowValueReached : in STD_LOGIC;
166                         -- buffer counter 0-6 used to buffer in the
                                beginning of a new row--
167                         bufReset              :       out STD_LOGIC;
168                         bufPause              :  out std_logic;
169                         bufLoadvalue :        out STD_LOGIC_VECTOR (3
                                downto 0);
170                         bufValReached:        in  STD_LOGIC;
171                          -- MEM Counters --
172                       --  MemRowCount : in STD_LOGIC_VECTOR (8 downto 0);
173                         ResetMemColumCount: out STD_LOGIC;
174                         MemColPause : out STD_LOGIC;
175                         ResetRowOffsetCounter : out STD_LOGIC;
176                         RowOffsetPause : out STD_LOGIC;
177                         -- write counter --
178                         Writereset : out STD_LOGIC;
179                         writepause : OUT STD_LOGIC;
180                         read_write: out std_logic; -- 1 read, 0 = write
181
182                         --
183                         done : out STD_LOGIC;
184                       --  stateout : out STD_LOGIC_VECTOR(3 downto 0); --);
                                -- for testing purpose only
185                       --  addressOut : out STD_LOGIC_VECTOR (31 downto 0));
186                        --- new in/outs due to extraction of FF ----
187                        next_state : out std_logic_vector(3 downto 0);
188                        state : in std_logic_vector(3 downto 0));
189                        --Muxctrl_in: in std_logic_vector(3 downto 0));
190 end component FSM;
191
192 component count4bit_latch is
```

```
193  Port (           Reset : in STD_LOGIC;
194                        CTRL : in  STD_LOGIC_VECTOR(1 downto 0);
195             lastval_in : in  STD_LOGIC_VECTOR (3 downto 0);
196             count_in : in  STD_LOGIC_VECTOR (3 downto 0);
197             lastval_out : out  STD_LOGIC_VECTOR (3 downto 0);
198             count_out : out  STD_LOGIC_VECTOR (3 downto 0);
199             targetR_in : in  STD_LOGIC;
200             targetR_out : out  STD_LOGIC);
201 end component count4bit_latch;
202
203 component counter_4bit is
204     Port ( --clk : in  STD_LOGIC;
205             resetToLoad : in  STD_LOGIC; -- active high
206                        load : in  STD_LOGIC_vector(3 downto 0);
207                        pause: in std_logic;
208                        count_in : in STD_LOGIC_VECTOR (3 downto 0);
209                        lastval_in : in STD_LOGIC_VECTOR (3 downto 0);
210                        ReachedTaget : out std_logic;
211                        lastval_out : out STD_LOGIC_VECTOR (3 downto 0);
212             count_out : out  STD_LOGIC_VECTOR (3 downto 0));
213 end component counter_4bit;
214
215 component counter9bit_latch is
216  Port (           CTRL : in  STD_LOGIC_VECTOR;
217             Reached_row_in : in  STD_LOGIC;
218             Reached_colum_in : in  STD_LOGIC;
219             Count_row_in : in  STD_LOGIC_VECTOR (8 downto 0);
220             Count_column_in : in  STD_LOGIC_VECTOR (8 downto 0);
221             Reached_row_out : out  STD_LOGIC;
222             Reached_column_out : out  STD_LOGIC;
223             count_row_out : out  STD_LOGIC_VECTOR (8 downto 0);
224             count_column_out : out  STD_LOGIC_VECTOR (8 downto 0));
225 end component counter9bit_latch;
226
227 component counter9bit is
228 Port ( --clk : in  STD_LOGIC;
229             resetToLoad : in  STD_LOGIC; -- active high
230                        load : in  STD_LOGIC_vector(8 downto 0); -- 9 bits
                                represent up to 512 dec
231                        pause: in std_logic;
232                        reset_column : in STD_LOGIC;
233                        Reached_row_in : std_LOGIC;
234                        Count_column_in : in STD_LOGIC_VECTOR (8 downto 0);
235                        Count_row_in : in STD_LOGIC_VECTOR (8 downto 0);
236                        ReachedTaget_column : out std_logic; --reached 0
237                        ReachedTaget_Row : out std_logic; -- reached 288
238             count_out_column : out  STD_LOGIC_VECTOR (8 downto 0);
239                        count_out_row : out  STD_LOGIC_VECTOR (8 downto 0)
240                        );
241 end component counter9bit;
242
243
244 component counter2bit_latch is
245 Port (    CTRL : in  STD_LOGIC_VECTOR(1 downto 0);
246             count_Low_in : in  STD_LOGIC_VECTOR (1 downto 0);
247             count_High_in : in  STD_LOGIC_VECTOR (6 downto 0);
248             count_Low_out : out  STD_LOGIC_VECTOR (1 downto 0);
249             Count_High_out : out  STD_LOGIC_VECTOR (6 downto 0));
250 end component counter2bit_latch;
251
252 component counter2bit is
253      Port ( --clk : in  STD_LOGIC;
254             reset : in  STD_LOGIC; -- active high
255                        pause : in STD_LOGIC;
256                        countLow_in : in STD_LOGIC_VECTOR (1 downto 0);
```

```
                          countHigh_in : in STD_LOGIC_VECTOR (6 downto 0);
           count_out_low : out  STD_LOGIC_VECTOR (1 downto 0);
                          count_out_high :out STD_LOGIC_VECTOR( 6 downto 0)
                          );
end component counter2bit;


component upcounter32bit_latch is
         Port ( CTRL : in  STD_LOGIC_VECTOR(1 downto 0);
           count_in : in  STD_LOGIC_VECTOR (31 downto 0);
           count_mem_in : in  STD_LOGIC_VECTOR (31 downto 0);
           count_out : out  STD_LOGIC_VECTOR (31 downto 0);
           count_mem_out : out  STD_LOGIC_VECTOR (31 downto 0));
end component upcounter32bit_latch;

component upcounter32bit is
    Port ( reset : in  STD_LOGIC;
           pause : in  STD_LOGIC;
                          count_in : in STD_LOGIC_VECTOR(31 downto 0);
           count_out : out  STD_LOGIC_VECTOR (31 downto 0);
                          count_out_mem : out  STD_LOGIC_VECTOR (31 downto 0)
                              );
end component upcounter32bit;

component Latch_control_top is
Port ( Reset : in  STD_LOGIC;
                          Req_in : in STD_LOGIC;
                          Ack_in : out STD_LOGIC;
                          Req_out : out STD_LOGIC;
                          Ack_out : in STD_LOGIC;
                          Master: out  STD_LOGIC;
                          Slave: out STD_LOGIC);
end component Latch_control_top;

component Fork is
    Port ( Req_in : in  STD_LOGIC;
           Req_out : out  STD_LOGIC_VECTOR (1 downto 0);
           Ack_out : in  STD_LOGIC_VECTOR (1 downto 0);
           Ack_in : out  STD_LOGIC);
end component Fork;

component Join is
    Port ( Req_in : in  STD_LOGIC_VECTOR (1 downto 0);
           Req_out : out  STD_LOGIC;
           Ack_out : in  STD_LOGIC;
           Ack_in : out  STD_LOGIC_VECTOR (1 downto 0));
end component Join;

----------------Test-------------
--------------------

begin

FSM_control : Latch_control_top port map(   Reset => NotReset,
                                                                                                                                  Req_in
                                                                                                                                      =>
                                                                                                                                      fs
                                                                                                                                      (0
                                                                                                                                      ,
                                                                                                                                  Ack_in
                                                                                                                                      =>
```

```
fsm_7Ac
(0)
,
Req_out
=>
fsm_rec
,
Ack_out
=>
fsm_ack
,
Master
=>
CTRL_FS
(0)
,
Slave
=>
CTRL_FS
(1)
)
;

FSM_fork : fork7port port map(Req_in => fsm_req_del,
           Ack_in => fsm_ack_out,
           Req_out => fsm_7Req,
           Ack_out => fsm_7Ack);

fsm_req_del <= fsm_req_out after 1us;
fsm_req <= fsm_7Req(1);
fsm_7Ack(1) <=fsm_ack ;

State_FF : FSM_latch port map ( CTRL => CTRL_FSM,
                                                          hold => hold,
                                                          reset => reset,
                                                          Next_state =>
                                                               int_sxt_state,
                                                          state => int_state,
                                                          muxCtrl =>
                                                               int_stateout,
                                                          stateout =>
                                                               int_mux_in  );

mxc : muxCTRL port map( next_state => int_state,-- int_sxt_state,
                                                             muxCtrl_in =>
                                                                  int_mux_in,
                                                             muxCtrl =>
                                                                  int_stateout)
                                                                  ;

statemachine : FSM port map (--clk => clk,
                                                                                  --
```

```
                                                                reset
                                                                =>
                                                                reset
                                                                ,
340                                                     start
                                                                =>
                                                                start
                                                                ,
341                                                     --
                                                            hold
                                                            =>
                                                            hold
                                                            ,
342                                                     regSelectReset
                                                                =>
                                                                regSel
                                                                ,
343                                                     regPause
                                                                =>
                                                                regPs
                                                                ,
344                                                     regLoad
                                                                =>
                                                                regL
                                                                ,
345                                             --
                                                    regValReached
                                                     =>
                                                    regVal
                                                    ,--
                                                    regisers
346                                                     bufReset
                                                                =>
                                                                bufSel
                                                                ,
347                                                     bufPause
                                                                =>
                                                                bufPs
                                                                ,
348                                                     bufLoadvalue
                                                                =>
```

```
                                                                    bufL
                                                                    ,
349                                                         bufValReached
                                                                    =>
                                                                    bufVal
                                                                    ,
                                                                    --
                                                                    buffer
350                                                         columnReset
                                                                    =>
                                                                    columnSel
                                                                    ,
351                                                         columnPause
                                                                    =>
                                                                    columnPs
                                                                    ,
352                                                         columnLoad
                                                                    =>
                                                                    columnL
                                                                    ,
353                                                         columnValueReached
                                                                    =>
                                                                    columnVal
                                                                    ,
                                                                    --
                                                                    column
354                                                         rowReset
                                                                    =>
                                                                    rowSel
                                                                    ,
355                                                         rowPause
                                                                    =>
                                                                    rowPs
                                                                    ,
                                                                    --
                                                                    not
                                                                    connected
356                                                         rowValueReached
```

```
                                                                    =>
                                                                    rowVal
                                                                    ,
                                                                    --
                                                                    row
357                                                         FIFOadress
                                                                    =>
                                                                    FIFOadress
                                                                    ,
358                                                         FIFOread
                                                                    =>
                                                                    FIFOread
                                                                    ,
359                                                         FIFOwrite
                                                                    =>
                                                                    FIFOwrite
                                                                    ,
360                                                         done
                                                                    =>
                                                                    done
                                                                    ,
361                                                         --
                                                                    stateout
                                                                    =>
                                                                    int_stateout
                                                                    ,
362                                                 --
                                                    MemRowCount
                                                     =>
                                                    MemRCount
                                                    ,
363                                                         ResetMemColumCount
                                                                    =>
                                                                    ResetMemCCount
                                                                    ,
364                                                         ResetRowOffsetCounter
                                                                    =>
                                                                    ResetOffsetCounter
                                                                    ,
365                                                         RowOffsetPause
                                                                    =>
```

```
                                                                    OffsetPause
                                                                    ,
366                                                              MemColPause
                                                                    =>
                                                                    MemColPause
                                                                    ,
367                                                              writereset
                                                                    =>
                                                                    Wreset
                                                                    ,
368                                                              writepause
                                                                    =>
                                                                    Wpause
                                                                    ,
369                                                              read_write
                                                                    =>
                                                                    read_write
                                                                    ,
370                                                              next_state
                                                                    =>
                                                                    int_sxt_state
                                                                    ,
371                                                              state
                                                                    =>
                                                                    int_state
                                                                    )
                                                                    ;
372                                                           --
                                                                 Muxctrl_in
                                                                 =>
                                                                 int_mux_in
                                                                 )
                                                                 ;
373
374  buf_Join : Join port map(Req_in(0) => buf_req_delay,
375                                                           Req_in
                                                                (1)
                                                               =>
                                                               fsm_7Req
```

```
(3),
Req_out => buf_req_in,
Ack_out => buf_ack_in,
Ack_in(0) => buf_ack_out,
Ack_in(1) => fsm_7Ack(3));
buf_req_delay <= buf_req_out after 1us;

buf_control : Latch_control_top port map( Reset => NotReset,
Req_in => bu
,
Ack_in => bu
,
Req_out => bu
,
Ack_out => bu
,
```

```
Master => CTRL_bu (0) ,
Slave => CTRL_bu (1) ) ;

bufFF : count4bit_latch port map ( Reset => reset,
CTRL => CTRL_buf ,
lastval_in => buf_lastval_in ,
count_in => buf_count_in ,
lastval_out => buf_lastval_out ,
count_out => buf_count_out ,
targetR_in => buf_target_in ,
targetR_out => bufVal ) ;
```

```
398
399 bufcounter : counter_4bit port map ( resetToLoad => bufSel,
400                                                                 load => bufL,
401                                                                 pause => bufPs,
402                                                                 ReachedTaget => buf_target_in,
403                                                                 count_in => buf_count_out,
404                                                                 count_out => buf_count_in,
405                                                                 lastval_in => buf_lastval_out,
406                                                                 lastval_out => buf_lastval_in);
407
408 reg_Join : Join port map( Req_in(0) => reg_write_req_out(0),
409                                                         Req_in(1) => fsm_7Req(2),
410                                                         Req_out
```

```
=> reg_req_in ,
Ack_out => reg_ack_in ,
Ack_in (0) => reg_write_ack_out (0) ,
Ack_in (1) => fsm_7ack (2)
) ;

reg_control : Latch_control_top port map(   Reset => NotReset ,
Req_in => reg_rec
,
Ack_in => reg_ack
,
Req_out => reg_rec
,
Ack_out => reg_ack
,
Master => CTRL_re
```

```
(0
,
Slave
=>
CT
(1
)
;

reg_fork: Fork port map( Req_in => reg_req_delay,
                         Req_out => reg_write_req_out,
                         Ack_out => reg_write_ack_out,
                         Ack_in => reg_ack_out);

reg_req_delay <= reg_req_out after 1us;
reg_req <= reg_write_req_out(1); --reg_req is output of component
reg_write_ack_out(1) <= reg_ack;

regFF : count4bit_latch port map (Reset => reset,
                                  CTRL => CTRL_reg,
                                  lastval_in => reg_lastval_in,
                                  count_in => reg_count_in,
                                  lastval_out =>
```

```
reg_lastval_out ,
    count_out => reg_count_out ,
    targetR_in => reg_target_in ,
    targetR_out => regVal ) ;

regcounter : counter_4bit port map ( resetToLoad => regSel ,
    load => regL ,
    pause => regPs ,
    ReachedTaget => reg_target_in ,
    count_in => reg_count_out ,
    count_out => reg_count_in ,
    lastval_in
```

```
                                        => reg_lastval_out ,
                              lastval_out => reg_lastval_in ) ;

Px_Join : Join port map(Req_in(0) => px_write_req_out(0),
                        Req_in(1) => fsm_7Req(4),
                        Req_out => px_req_in,
                        Ack_out => px_ack_in,
                        Ack_in(0) => px_write_ack_out(0),
                        Ack_in(1) => fsm_7Ack(4));

px_control : Latch_control_top port map(    Reset => NotReset,
                                            Req_in => px
                                            ,
```

```
Ack_in => px_ack_,
Req_out => px_req_,
Ack_out => px_ack_,
Master => CTRL_co(0),
Slave => CTRL_co(1) );

Px_fork: Fork port map( Req_in => px_req_delay,
    Req_out => px_write_req_out,
    Ack_out => px_write_ack_out,
    Ack_in => px_ack_out );

px_req_delay <= px_req_out after 1us;
px_req <= px_write_req_out(1);
```

```
472 px_write_ack_out(1) <= px_ack ;
473
474 pxFF    :       counter9bit_latch port map ( CTRL => CTRL_counter9bit,
475                                                                    Reached_row_in
                                                                           =>
                                                                           temp_reached_row_in
                                                                           ,
476                                                                    Reached_colum_in
                                                                           =>
                                                                           temp_reached_column_in
                                                                           ,
477                                                                    Count_row_in
                                                                           =>
                                                                           temp_count_row_in
                                                                           ,
478                                                                    Count_column_in
                                                                           =>
                                                                           temp_count_column_in
                                                                           ,
479                                                                    Reached_row_out
                                                                           =>
                                                                           temp_reached_row_out
                                                                           ,
480                                                                    Reached_column_out
                                                                           =>
                                                                           temp_reached_column_out
                                                                           ,
481                                                                    count_row_out
                                                                           =>
                                                                           temp_count_row_out
                                                                           ,
482                                                                    count_column_out
                                                                           =>
                                                                           temp_count_column_out
                                                                           )
                                                                           ;
483
484 pxcounter : counter9bit port map(resetToLoad => columnSel,
485                                                                  reset_column
                                                                         =>
```

```
                rowSel
                ,
486             load
                =>
                columnL
                ,
487             pause
                =>
                columnPs
                ,
488             ReachedTaget_column
                =>
                temp_reached_column_in
                ,
489             ReachedTaget_row
                =>
                temp_reached_row_in
                ,
490             count_out_column
                =>
                temp_count_column_in
                ,
491             count_out_row
                =>
                temp_count_row_in
                ,
492             Reached_row_in
                =>
                temp_reached_row_out
                ,
493             Count_column_in
                =>
                temp_count_column_out
                ,
494             Count_row_in
                =>
                temp_count_row_out
```

```
)
;

offset_Join : Join port map(Req_in(0) => offset_write_req_out(0),
    Req_in(1) => fsm_7Req(5),
    Req_out => offset_req_in,
    Ack_out => offset_ack_in,
    Ack_in(0) => offset_write_ack_out(0),
    Ack_in(1) => fsm_7ack(5)
    );

offset_control : Latch_control_top port map( Reset => NotReset,
    Req_in => of
    ,
    Ack_in => of
    ,
    Req_out
```

```
=> offset_
,
Ack_out => offset_
,
Master => CTRL_co
(0)
,
Slave => CTRL_co
(1)
)
;
offset_fork : Fork port map( Req_in => offset_req_delay,
                             Req_out => offset_write_req_out,
                             Ack_out => offset_write_ack_out,
                             Ack_in => offset_ack_out );

offset_req_delay <=     offset_req_out after 1us;
offset_req <= offset_write_req_out(1);
offset_write_ack_out(1) <= offset_ack;

offset_FF : counter2bit_latch port map ( CTRL => CTRL_counter2bit,
                                        count_Low_in => temp_count_low_in,
```

```
                                        count_High_in => temp_count_high_in ,
                                        count_Low_out => temp_count_low_out ,
                                        Count_High_out => temp_count_high_ou) ;

offsetcounter : counter2bit port map ( reset => ResetOffsetCounter,
                                        pause => OffsetPause ,
                                        count_out_low => temp_count_l ,
                                        count_out_high => temp_count_hi ,
                                        countLow_in => temp_count_l ,
                                        countHigh_in => temp_count_h ) ;

Write_join : Join port map(Req_in(0) => Fork_write_req_out(0),
                           Req_in(1) =>
```

```
fsm_7Req
(6)
,
Req_out
=>
Write_req_in
,
Ack_out
=>
Write_ack_in
,
Ack_in
(0)
=>
Fork_write_ack_out
(0)
,
Ack_in
(1)
=>
fsm_7ack
(6)
)
;

Write_control : Latch_control_top port map( Reset => NotReset,
Req_in
=>
Write_r
,
Ack_in
=>
Write_a
,
Req_out
=>
write_r
,
Ack_out
=>
```

```
wr
,
Master
=>
CT
(0
,
Slave
=>
CT
(1
)
;


write_fork : Fork port map( Req_in => Fork_req_delay ,
Req_out
=>
Fork_write_req_out
,
Ack_out
=>
Fork_write_ack_out
,
Ack_in
=>
write_ack_out
)
;

Fork_req_delay <= write_req_out after 1us;
write_req <= Fork_write_req_out(1);
Fork_write_ack_out(1) <= write_ack;

writecounter_FF :       upcounter32bit_latch port map(  CTRL =>
     CTRL_upcount32bit ,
```

```
writecounter : upcounter32bit port map ( reset => Wreset,
    pause => Wpause ,
    count_in => Wcount_out ,
    count_out => Wcount_in ,
    count_out_mem => Wcount_mem_in ) ;


---------------TEST------------------------------------------


------------------------------------------------------------------
--NotReset <= not(reset);
NotReset <= reset;
-- Buf counter signal--
stateout <= int_stateout;

-- reg counter signal--
regcount <= reg_count_out;

---px counter 9 bit FFout --
Rowcount <= temp_count_row_out;
PxNr_out <= temp_count_column_out;
rowVal <= temp_reached_row_out;
columnVal <= temp_reached_column_out;
--------------------------------------
----------- offset counter -----------
OffsetOut <= temp_count_low_out;
```

```
591 MemColumCountOut <=temp_count_high_out;
592 end Behavioral;
```

Bibliography

[1] K. Killpack F. Dartu C.S Amin, N. Menezes. Stastistic simple timing analysis: how simple can we get? 2005.

[2] Luciano Lavagno Jordi Cortadella. Desynchronization: synthesis of asynchronous circuits from sunchronous specifications. *IEEE Transactions on Computer Aided design og intergraded circuits*, 25(10), 2006.

[3] et al. Josep Carmona, Jordi Cortadella. Elastic circuits. *IEEE Transactions on Computer Aided design og intergraded circuits*, 28(10), 2009.

[4] Davide Pandini Christos P. Sotiriou Nikolas Andrikos, Luciano Lavagno. A fully automated desynchrononization flow for synchronous circuits. 2007.

[5] Martin Simlastik and Viera Stopjakova. Automated synchronous-to-asynchronous circuits conversion: A survey. pages 348–358, 2008.

[6] Jens Sparsø. *Asynchronous Circuit Design - A tutorial*. Technical University of Denmark, 2006.

[7] Paul Day Steven Furber. four-phase micropipeline latch control circuits. page 8.

[8] Peter A. Beerel Ayoob E. Dooply Steven M. Nowick, Kenneth Y. Yun. Speculative completion for the design of high-performance asynchronous dynamic adders. 1997.