Examensarbete

Asynchronous Wrapper for Globally Asynchronous Locally Synchronous Systems

Olof Manbo

Reg nr: LiTH-ISY-EX-3210-2002
2002-06-06

Asynchronous Wrapper for Globally Asynchronous Locally Synchronous Systems

Examensarbete utfört i Elektroniksystem vid Linköpings tekniska högskola

av
Olof Manbo

Reg nr: LiTH-ISY-EX-3210-2002

Handledare: Kent Palmkvist
Examinator: Kent Palmkvist

Linköping 2002-06-06

Avdelning, Institution
Division, Department

Institutionen för Systemteknik
581 83 LINKÖPING

Datum
Date
2002-06-06

Språk
Language
Svenska/Swedish
X Engelska/English

Rapporttyp
Report category
Licentiatavhandling
X Examensarbete
C-uppsats
D-uppsats
Övrig rapport

ISBN

ISRN LITH-ISY-EX-3210-2002

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2002/3210/

Titel Asynkron wrapper för globalt asynkrona lokalt synkrona system
Title

Asynchronous Wrapper for Globally Asynchronous Locally Synchronous Systems

Författare Olof Manbo
Author

Sammanfattning
Abstract

This thesis is investigating the new globally asynchronous locally synchronous (GALS) technology for integrated circuits. Different types of asynchronous wrappers are tested and a new wrapper design is presented. It also investigates the possibility to use VHDL for asynchronous simulation and synthesis. The conclusions are that the GALS technology is possible to use but that it needs new synthesis tools, because todays tools are designed for synchronous technology.

Nyckelord
Keyword
GALS, asynchronous, wrapper, electronics, VHDL, FPGA, VLSI

Abstract

This thesis is investigating the new globally asynchronous locally synchronous (GALS) technology for integrated circuits. Different types of asynchronous wrappers are tested and a new wrapper design is presented. It also investigates the possibility to use VHDL for asynchronous simulation and synthesis. The conclusions are that the GALS technology is possible to use but that it needs new synthesis tools, because todays tools are designed for synchronous technology.

Table of contents

1 Introduction

This project examines the possibility of using asynchronous signaling between different clocked parts of an integrated circuit. The idea is to feed every different part of the chip with it's own clock, which decreases the problem with clock skew and makes it easier to have different clock-frequencies on the same chip. It should also be possible to have storage elements between the different processing elements, and these should be easier to implement in asynchronous technology.

The idea behind this is that it will in the future be possible to have a whole system on one chip, which will imply that there is more than one clock frequency per chip. There must exist a solution to this problem before so called System-On-Chip-ware can become possible.

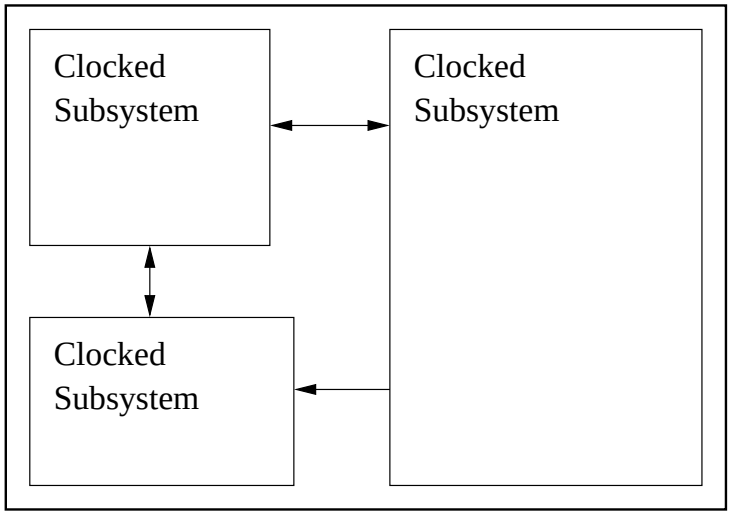


Fig 1. Example of an integrated circuit with several clocks. On a real system-on-chip there will also usually be analog parts, but this thesis only covers communication between the digital parts.

1.1 Goal

The goal of this project is to design a so called asynchronous wrapper in VHDL and test the functionality in an FPGA. It should also be possible to have a FIFO, a storage element, between the wrappers. It should be very interesting to see if this works because the synthesis tool is designed for clocked circuits.

1.2 Programs and hardware

The computer used for this thesis was a Sun Ultra 10 with Solaris 7. For VHDL programming, simulation and synthesis FPGA Advantage 5.2 from Mentor Graphics was used. FPGA Advantage consists of HDL Designer 2001.5, ModelSim 5.5e and LeonardoSpectrum v2001_1d.45. For FPGA implementation Xilinx Design Manager 4.1.01i was used. The hardware used for FPGA implementation was the XS40-10XL+ board from Xess Corporation which has an XC4010XL FPGA from Xilinx. The report was written using Adobe FrameMaker 5.5.

2 Technology introduction

Since the used technologies are not the most common in todays electronic design, a brief introduction to those technologies will be made. Asynchronous circuit design is actually an old technology, even older than synchronous circuit technology. The clocked technology later became standard and asynchronous design have never been able to compete with clocked circuits until recently. The new interest in asynchronous circuits comes from the fact that new research has simplified the design of them and that synchronous design now has encountered problems that perhaps can be solved more easily by using asynchronous circuit design. As the project also tries to combine asynchronous and synchronous technology you can combine the two paradigms to get the benefits from both.

2.1 Asynchronous circuits

When students are studying switching theory, they are usually taught that they should never bother to design an asynchronous circuit [2]. This might be the right way to think when designing simple circuits with TTL gates and stuff like that, but recently asynchronous design methodologies have become interesting again [1]. The problem with clock-skew and the ever increasing number of transistors on a single chip might be easier to handle with asynchronous or mixed synchronous and asynchronous techniques. In a clocked circuit everything will happen at the same time, when the clock rises, which will make the power consumption very high at the rising edge of the clock. If there is no clock, or if there is a lot of different clocks, the power consumption will be more spread out over time. The speed of a synchronous circuit will always be as slow as the slowest part, as the clock-frequency has to be chosen according to the slowest part. In an asynchronous circuit every part can work at its own maximum speed. A disadvantage with asynchronous circuits is that you cannot use dynamic logic circuits, because they use clocking to improve performance.

The biggest problem with asynchronous design is the fact that the circuits has to be hazard-free, which makes it harder to design the logical nets. An example of a hazard is when a logical net has a low output, and then the inputs changes, but the output should still be low. If the logic is not designed to be hazard free the output might go high for a short while before it stabilizes at a zero. When a clock is used this will not be a problem if the clock period is long enough for the circuit to stabilize, but

for asynchronous systems this could be a disaster. Fortunately synthesis tools exist, although the asynchronous paradigm hasn't become big enough for any commercial program to exist.

When asynchronous technology is used, sometimes not only the levels on the signals are of interest, but also the transitions between the levels. To show this plus and minus signs are used, i. e., a+ means that the signal a goes from low to high, and a- means that the signal goes from high to low.

2.2 Communication protocols

Since there is no clock in asynchronous circuits, data has to be sent with extra control signals called req, for request and ack, for acknowledge (see fig 2). Usually a four-phase protocol is used where req goes up, followed by ack and then req goes down, followed by ack. The data should be valid between req going to one and ack returning back to zero. Of course the signals ack+ and req- doesn't matter and that is why a two-phase protocol using transition signalling might be preferred. Transition signalling differs from the "normal" signalling in that the level of the control signals has no meaning. Instead the only thing that matters is when the signal changes. This means that a rising edge is equivalent to a falling edge. These changes are called events. When transition signalling is used for the communication protocol it means that there is an event on req and then ack answers with an event. The data should be valid between the events.

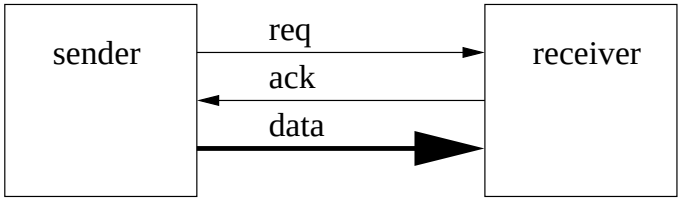


Fig 2. Asynchronous circuit sending data.

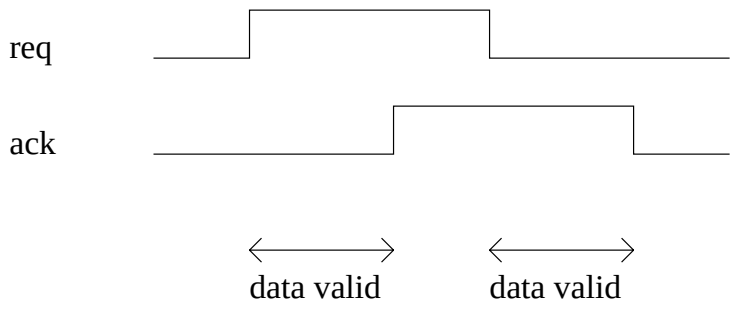


Fig 3. Two-phase protocol.

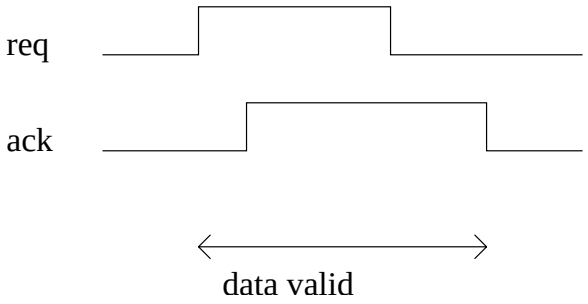


Fig 4. Four-phase protocol.

2.3 Hazard-free circuits

There exist two models for asynchronous circuits, Huffman circuits and Muller circuits. Huffman circuits are also called delay-insensitive circuits. This is because the circuits are guaranteed to work regardless of gate and wire delays, as long as the bound on the delay is known. The easiest way to make a Huffman circuit is when you only let one input change at a time. This is called single-input-change. Usually this is to restrictive so multiple-input change circuits is a better way to design the circuit. The latest way to synthesize a multiple-input change Huffman circuit is to use something called extended burst-mode. Extended burst-mode also puts some restrictions on the circuit, but is still a lot more flexible than single-input change.

Muller circuits are also called speed-independent. This model is hazard free under the assumption that the gate delay is unbounded but finite, and that there is no delay on wires.

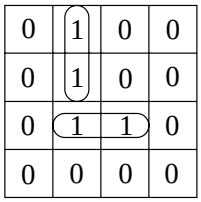


Fig 5. Karnaugh map of circuit with risk of 1-1 hazard.

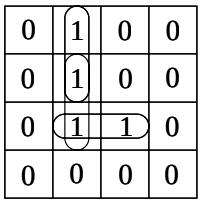


Fig 6. Karnaugh map of circuit without risk of 1-1 hazard.

Now an example of a hazard in a single-input circuit will be shown. Consider the karnaugh maps in fig 5 and fig 6. In fig 5 the ones are put into two groups. This is the normal way to do it if a synchronous circuit is designed, but the problem is that the circuit can go from one group to the other and the output should still be one. Because the transition is between the groups, the circuit may produce a hazard. In fig 6 an extra group is added in the karnaugh map. Then, independent of the input change, the transition will always be inside a group, and thus there will be no hazards, at least between inputs where the output should be high.

This was a simple example of a hazard and hazard free circuit, but today there exists no simple solution to the synthesis of hazard-free circuits. For small synchronous nets, karnaugh maps are an excellent way to do the synthesis “by hand”, but even for small asynchronous nets you need to use rather complicated algorithms is needed, which implies the use of synthesis tools.

2.4 Extended burst-mode

A normal FSM works as shown in fig 7. The states change when they have the correct value. In the example the machine will go from state zero to state one when the signal a is one.

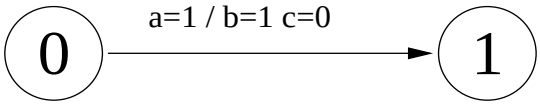


Fig 7. Example of a FSM transition. The signal a is an input signal, b and c are outputs.

The most popular model for asynchronous finite state machines (AFSM) is called Extended Burst Mode (XBM). In a normal FSM:s the values of the signals decide when it should go to a new state. When using XBM the transitions of the signals is what is important. When a normal FSM needs something like a=1 to change state, XBM uses the transition a+. At least one input signal should change between every state, since the lack of a clock makes it impossible to stay in a state without having to wait for a signal change (a synchronous circuit could be designed using XBM if the clock is treated as an input signal). XBM also allows something called directed don't cares, which allows a signal to either keep the same value or change once. Of course another signal must have a "normal" transition since a signal that only may change cannot be used to determine what state the AFSM is in. A directed don't care also has to be followed by a "normal" transition on the same signal. If directed don't cares are not allowed the model is just called Burst Mode (BM).

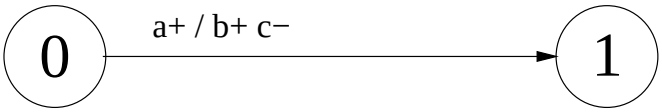


Fig 8. Example of an XBM transition working in the same way as the FSM in fig 7. The signal a is an input signal, b and c are outputs. A plus means that the signal should go high, a minus that it should go low. If there is an asterisk after the signal name, instead of a plus or minus, the signal is a directed don't care.

2.5 Metastability

A big problem with the asynchronous paradigm is the communication between asynchronous and synchronous circuits. In fig 9 a is simple approach to synchronize an asynchronous signal shown. Usually this works fine. When the input to the D-flip-flop changes before the rising edge of the clock the output will only change when the clock rises, and will then be synchronized. The problem occurs when the input changes too close in time to the rising edge of the clock. The flip-flop won't know if it should change or not, and will enter a metastable state, with the output not being high nor low. One might then say that this is not a big problem, the flip-flop will enter the correct state next clock cycle, but the problem is that theoretically the circuit could stay in the metastable state forever. If you put more D-flip-flops in series the risk of entering the metastable state will be lower, but as long as you do it this way preventing metastability can never be 100 percent guaranteed. With many flip-flops in a row latency will also increase. But there exist another solution to this problem. As shown later, asynchronous wrappers are designed to prevent metastability from ever occurring.

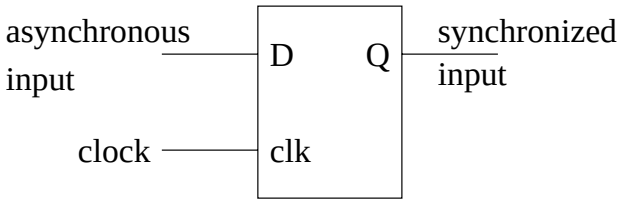


Fig 9. Simple circuit for synchronization.

2.6 Globally Asynchronous Locally Synchronous

The idea of Globally Asynchronous Locally Synchronous (GALS) circuits is that it is possible to get some of the advantages of asynchronous technology without having to throw away all the knowledge in synchronous design. The GALS paradigm uses asynchronous signalling between different clocked units on a chip. Every clocked part is surrounded by an asynchronous wrapper that takes care of the signalling between the different clocked parts. Since the design of the asynchronous wrapper is much easier than designing a whole asynchronous system and the problem of clock-skew is much smaller some of the benefits of asynchronous design are

gained without having to get rid of the D-flip-flops. Different blocks can also use different clock-frequencies. This could also decrease the design time, since parts of the circuit that are not used too often can use a lower clock frequency and don't have to be optimized as much as the often-used parts. If the clock period is optimized to be as long as possible for every part, power consumption will also be lower, as every part can work at as low clock frequency as possible. Since a global clock creates a lot of noise by having a spike in the power spectrum, problems with noise will also decrease.

2.7 Muller-C elements

A Muller-C element is an important gate in asynchronous design. It has a zero on the output if both inputs are zero, and a one if both inputs are one but keeps it's value otherwise. This could sometimes be important to make hazard-free circuits because it changes value only after every input signal have changed. It could also be used as an AND-gate for events if transition signalling is used. The OR-function is then provided by an XOR-gate [4].

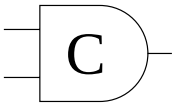


Fig 10. Symbol of Muller-C element.

2.8 FIFO

FIFO stands for First In-First Out and that describes how a FIFO works. It is important to recognize that a FIFO is not similar to a shift-register since the length of a FIFO is dynamic. A four bit long shift-register will always have to shift four times to get the latest input to the output, but a FIFO can be empty or only store data in a few of its storage elements. This means that if the FIFO is empty the latest input can be read at the output directly.

2.9 Micropipelines

A micropipeline [4] is an asynchronous pipeline that could be used to make pipelined computations. If the pipeline only has storage-elements without any logic, it will work like a FIFO, which is the way it has been used in this thesis. The control-circuit in the micropipeline uses transition signalling and that means that it uses a two-phase protocol. The micropipeline makes extensive use of Muller-C elements.

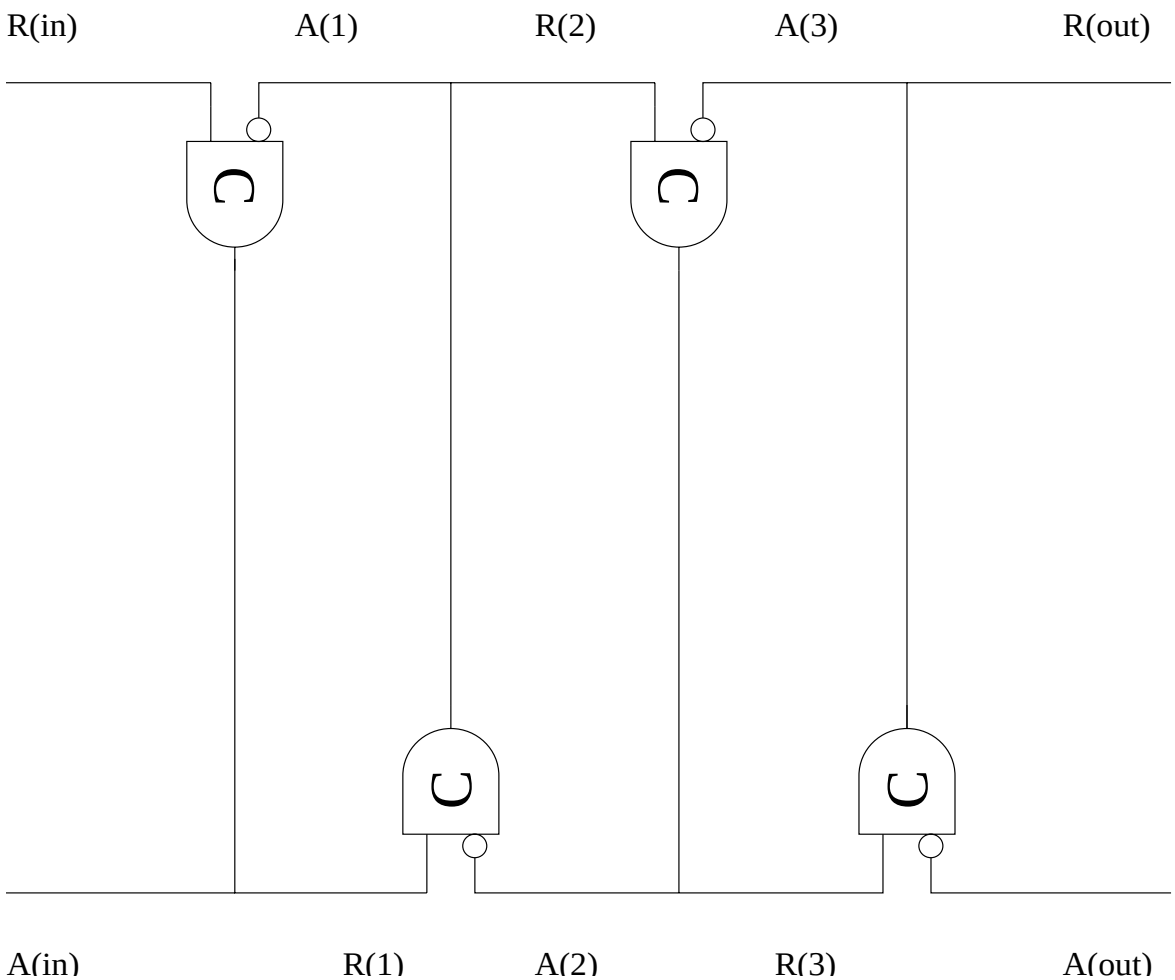


Fig 11. The control circuit of the Micropipeline. The outputs from the Muller-C elements control the storage elements. A stands for ack and R stands for req. Since there are four Muller-C elements, this control circuit is designed for a 4 bit long FIFO.

2.10 Synthesis tool for extended burst-mode

The 3D synthesis tool has been used to create the nets for the in- and outport. It is very simple to use, the XBM AFSM should be defined as in fig 12. The result is shown in fig 13. A more detailed text about the tool can be found in [3].

```
input enable 0
input ack 0
output stretch 0
output req 0

0 1 enable+| stretch+ req+
1 2 ack+| stretch-
2 3 enable-| stretch+ req-
3 0 ack-| stretch-
```

Fig 12. Example of input file to the 3D synthesis tool. This XBM-spec is outport.

```
stretch =
  enable’ ack +
  enable ack’

req =
  enable
```

Fig 13. Example of equation file from the tool. These are the equations of outport.

Another synthesis program called Minimalist [8] was also tested, but that program could only handle burst-mode specifications, not XBM. This was unfortunate, because that program had some nice features such as the possibility to show the logical nets graphically. But when the program to use was selected it was unknown that the designed AFSM would only need burst-mode.

3 Introduction to asynchronous wrappers

The idea of an asynchronous wrapper is that it is used as a camouflage to hide the fact that it is clocked on the inside. That means that a clocked circuit could be used inside the wrapper, but on the outside it acts like an asynchronous circuit. The maybe most important thing needed to accomplish this is the stretchable clock. This clock acts like a normal clock if it is not required to stretch, but when the clocked circuit needs an input or it has to output some data the clock stretches the low part of the clock-period. This means that the clocked circuit sleeps when it is waiting for new data or for outputting data. Communication between wrappers has to be controlled by a handshake protocol since there is no global clock. Previous designs of wrappers have used a four-phase protocol for global communication, the design in this thesis uses a two-phase protocol, since it has to work with micropipelines.

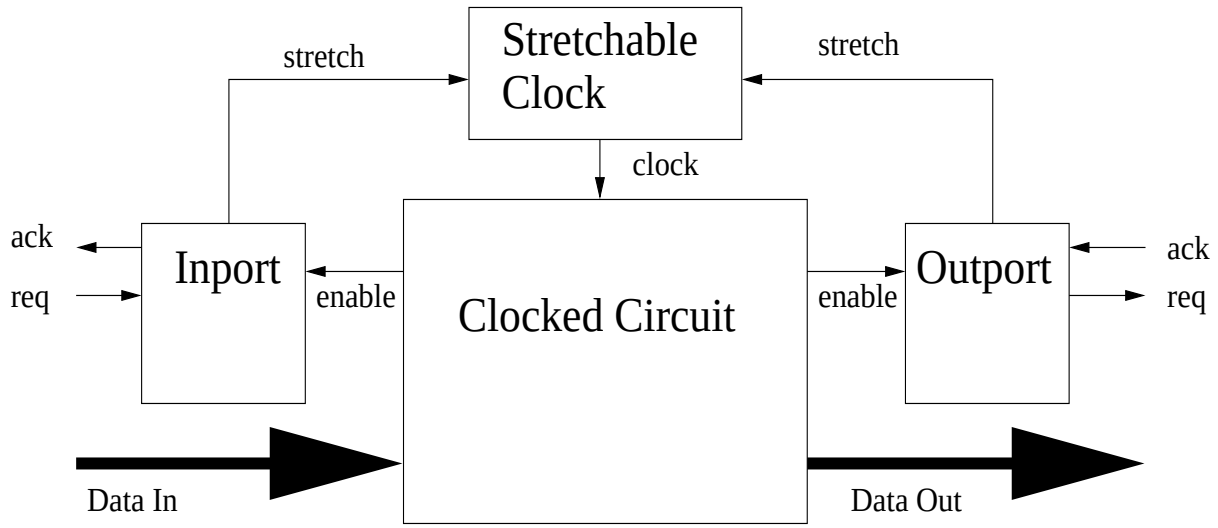


Fig 14. Principal design of an asynchronous wrapper. Inport is the control circuit for data input, Outport controls the data output.

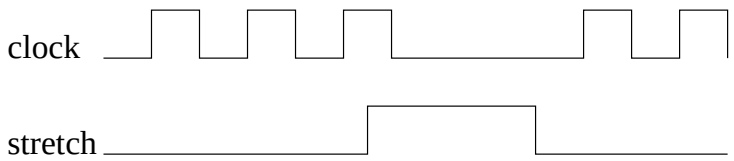


Fig 15. Example of the output from a stretchable clock. Notice that only low period of the clock is stretched, even if stretch goes high when the clock is high. Since the clock sleeps when the circuit is not active, power consumption will be lower.

3.1 Preventing metastability

The metastability problem may occur if you synchronize an asynchronous signal, as mentioned above. The asynchronous wrapper solves this problem by doing it the opposite way. Since asynchronous communication only takes place when the clock stretches, the signals will be stable when the clock starts again, and thus there is no risk of synchronization failure. The idea is that instead of synchronizing the asynchronous signals, the synchronous parts are “unsynchronized”. A problem with this solution may arise when a chip communicates asynchronously with other clocked chips. In this situation there will be problems with synchronization anyway, but if you are planning to have a whole system on one custom designed integrated circuit this will not be a problem.

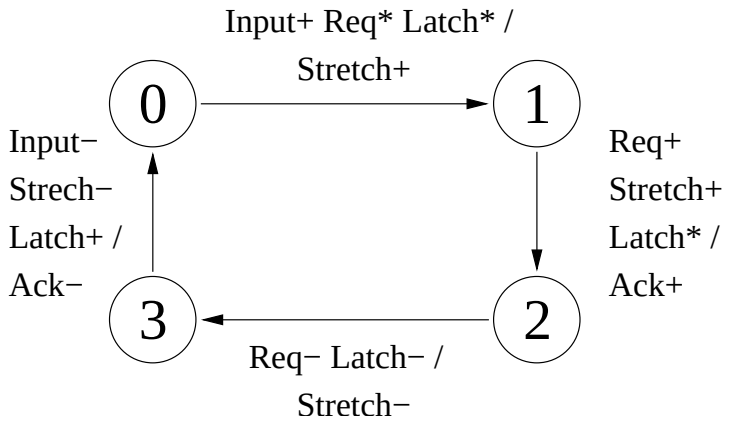
3.2 Previous designs

When this project was started, the first paper examined was [5] which described a unit for asynchronous communication between different locally synchronous blocks. This was an example of an asynchronous wrapper that included a FIFO. It also had a data-bus that worked in both directions. The FIFO was implemented via a RAM. This made the unit rather complicated with a lot of communication between the synchronous and asynchronous parts of the unit. Therefor was a search for other papers about asynchronous wrappers done. First found was a paper written by Bormann and Cheung [6], which didn’t have a FIFO but introduced the concept of stretchable clocks. A paper by Muttersbach et. al. [7] was also examined. It proposed a similar design but also introduced the concept of transition signalling on the enable signal, which made the design a bit simpler, but they still used a four-phase

protocol which made the solution a bit complicated. The final design also uses micropipelines that was introduced by Sutherland [4].

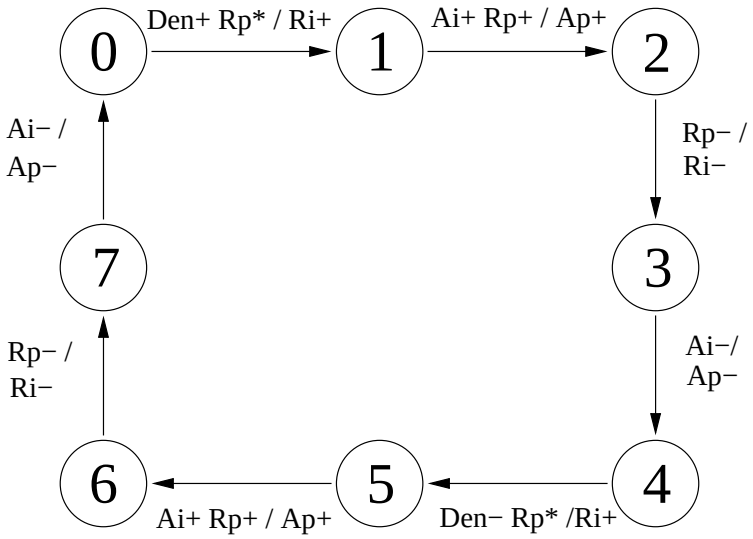
3.3 The control circuit

The Bormann/Cheung wrapper has a control unit AFSM specification as shown in fig 16. The design has only four states, but there is a lot of signals, and the signal stretch is used both as an output and an input. The big problem is that the signal input does not use transition signalling, which means that the clock needs to have a rising edge between state three and zero. Because input is sent from the clocked circuit it cannot change unless the clock is ticking. This means that one extra clock cycle is needed every time data is sent or received. If data should be transmitted every clock cycle this is a huge drawback. The most important thing about this wrapper is that it introduced the idea of a stretchable clock, which is what makes asynchronous wrappers possible in the first place.



$$Ack = Req*Stretch + Input*Ack + \overline{Latch}*Ack$$
$$Stretch = Req*Input + Input*\overline{Ack} + Latch*Stretch$$
$$Latch = \overline{Req} * Ack$$

Fig 16. The inport of the Bormann/Cheung wrapper, XBM specification and boolean equations. The signal Input is equivalent to enable. Latch controls the input latches of the wrapper, which are needed since this design does not use a FIFO.

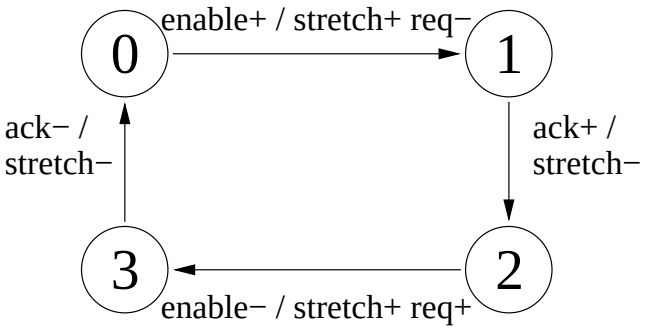


$$Ri = Rp*Ri + \overline{Den}*Z0 + Den*\overline{Ap}*\overline{Z0}$$
$$Ap = Rp*Ai + Ai*Ap$$
$$Z0 = \overline{Rp}*Z0 + \overline{Ai}*Z0 + Den*\overline{Rp}*Ap$$

Fig 17. XBM specification and Boolean equations of the input port of the Muttersbach et. al. wrapper. Note that the equations are almost as simple as they are for the Bromann/Cheung wrapper despite this wrapper having 8 states. Den are the enable signal, Rp stands for req, Ri is stretch, Ap is ach and Ai is an acknowledge signal from the stretchclock.

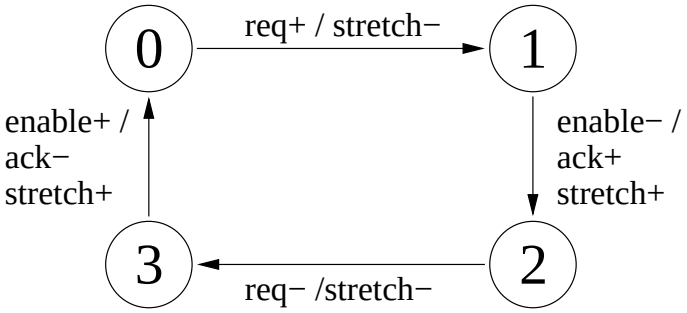
The Bormann/Cheung wrapper was improved by Muttersbach et. al. which made a control circuit according to fig 17. They introduced transition signalling on the enable signal, which simplified the design because no output signal needed to be used as an input. On the other hand the AFSM needed 8 states, but the control logic was not much more complicated, and the wrapper could transmit data every clock cycle.

4 Designing the wrapper



$$req = enable$$
$$stretch = \overline{enable}*ack + enable*\overline{ack}$$

Fig 18. XBM specification and boolean equations of outport.



$$ack = \overline{enable}$$
$$stretch = enable*\overline{req} + \overline{enable}*req$$

Fig 19. XBM specification and boolean equations of inport. Notice that enable should be high when the circuit is started.

The biggest inspiration to the new wrapper design came from the Muttersbach et. al. wrapper. The Liljeberg et. al. wrapper described the idea of using a FIFO, but when a micropipeline FIFO was connected between the wrappers the new design became much more simple than the Liljeberg et. al. wrapper. The new design of the wrapper differs from Muttersbach et. al. only in the way the AFSM control circuits are designed and how they behave. It uses transition signalling on every signal which

made it possible to have only four states. The signalling is now so simple that no feedback is needed. Ironically, after a lot of studies of AFSM:s it is enough to use simple combinatorial logic, but it is still important to make sure that the circuit is hazard-free since it is still working in an asynchronous environment.

The outport starts at state zero waiting for the synchronous circuit to set enable high. When enable goes high, the correct values should be on the output of the synchronous circuit, and thus req goes high. Stretch also goes high because the circuit should not to do anything before the output data has been read. At state one the circuit waits for ack to go high, which indicates that the data has been read so it is time to set stretch low and wait for another event on enable, which is done at state two. The rest of the states work the same way, but with negative transitions except on stretch.

The inport starts with enable set high because there should always be a req transition before there is a transition on ack. Otherwise it works like the outport but reads the data instead of outputting it.

The disadvantage with this design is that it will not work without a FIFO, because then the outport will have stretch high until ack changes, and thus the synchronous logic will not work until the inport is ready. This problem could be overcome by always having a FIFO between the ports, even if it is only one cell long. The previous designs of wrappers uses latches that store the values between the ports so this should not make the performance of the design worse than the previous designs. A circuit like the one in fig 20 can be used to generate a stretchable clock using a external clock source, but this circuit has the disadvantage that there will always be one rising edge of the clock after stretch goes high.This can of course be taken care of in the synchronous circuit, but this will have the disadvantage that data can only be sent or received every other clock cycle instead of once every clock cycle. When the circuit was synthesized into an FPGA it used an external clock, but if a custom ASIC is used an internal ring oscillator is a better choice.

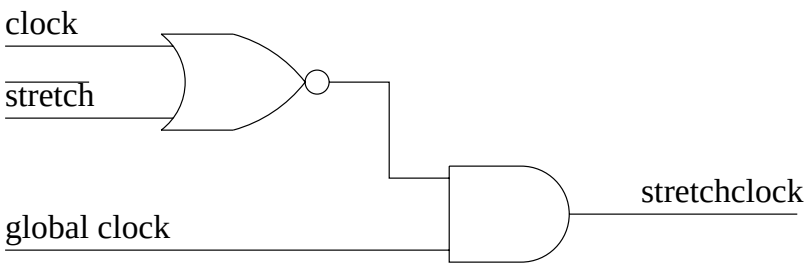


Fig 20. A way to make a stretchable clock using external clocks.

5 Implementing the wrapper

5.1 Tools

The tool used for VHDL coding and compilation was FPGA Advantage from Mentor Graphics. It consists of HDL Designer, ModelSim and LeonardoSpectrum. HDL Designer makes it simple to use a top-down methodology when programming VHDL. Although this sounds very good it was sometimes very frustrating to use this program, as the graphic user interface didn't always work as expected.

Modelsim is the simulation program, and it is easy to use and did not give much trouble.

LeonardoSpectrum is the program used for synthesis and it was rather tricky to get it to work. The synthesis result depended on the order of the input files, which could make sense if there was a special order that the files should be in. The strange thing is that LeonardoSpectrum is started from HDL designer, and the input files is loaded automatically. Still, the files was read in the wrong order. A solution to this problem has not been found.

For FPGA implementation Xilinx Design Manager was used, which used the output files from LeonardoSpectrum, and there was no problems with it.

5.2 VHDL-programming

After a lot of literature studies a behavioral description in VHDL was created using mentor graphics FPGA Advantage. First a wrapper as described Bormann and Cheung was created. This was a simple task as the equations and everything that was needed to make the wrapper was available. After the circuit had been simulated I became aware of the problems mentioned above regarding this wrapper became apparent, so implementation of the Muttersbach et. al. wrapper was started. It also worked fine, but then A FIFO also had to be implemented. This proved to be difficult, since the paper by Liljeberg et. al. didn't give as much information on how to implement their wrapper. Their wrapper was also much more complicated, so a lot of effort was put into finding a way to simplify their design. The biggest problem was to implement the FIFO-design, since it was based on the use of a memory. One solution was found, but then micropipelines was introduced. Micropipelines was

not only a very good solution to make a FIFO, it also introduced transition signalling, which allowed for simplifications of the control circuits of the wrapper. As previously mentioned, the end result was combinatorial logic without the need for feedback, which should make the wrapper very fast and reliable. The micropipeline also had another advantage. The storage elements in the pipeline will also work as amplifiers, which will remove the need for extra amplifiers on long bus lines.

5.3 Simulation

Simulation of asynchronous and GALS circuits was not to complicated The compiler had no problem with the lack of a clock. In fig 21 there is an example of a block diagram from HDL Designer showing an AFSM. An extra delayblock was needed to control the feedback as the output signals needed to be fed back to the AFSM. This showed an disadvantage of VHDL if it should be used for asynchronous design. But this was the only technical problem I encountered when compiling and simulating the circuit and it had an simple solution. So for simulation of GALS circuits VHDL was very useful.

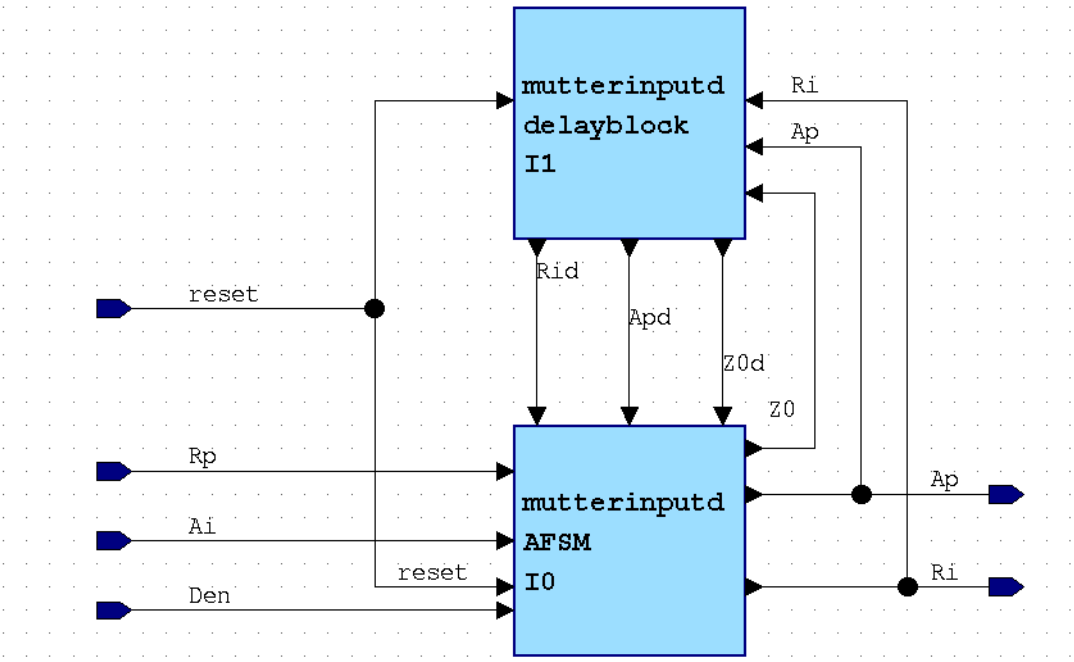


Fig 21. Example of an AFSM in HDL Designer. The AFSM block is the logical nets end delayblock is used for feedback of the output and internal state signals.

The first circuit that was simulated was the simple circuit shown in fig 22. It made it possible to see if the stretch signal worked as it should. In fig 23 there is an example of a simulation of this circuit. First are four bits loaded into the micropipeline and then they are outputted. The only signals forced manually in the simulation is enableo and enablei. Notice that stretchi starts high, and goes low when there is data to read on the output of the micropipeline. The glitches on stretcho are there because the AFSM needs to go from state 0 or 2 to 1 or 3 to change req, even if ack answers immediately. If the stretchable clock is designed correctly these short stretch signals will not affect the clock frequency. It is also important to notice the transition signalling, i. e., the circuit is not triggered by the rising edge on the enable signals, but every time the signals change.

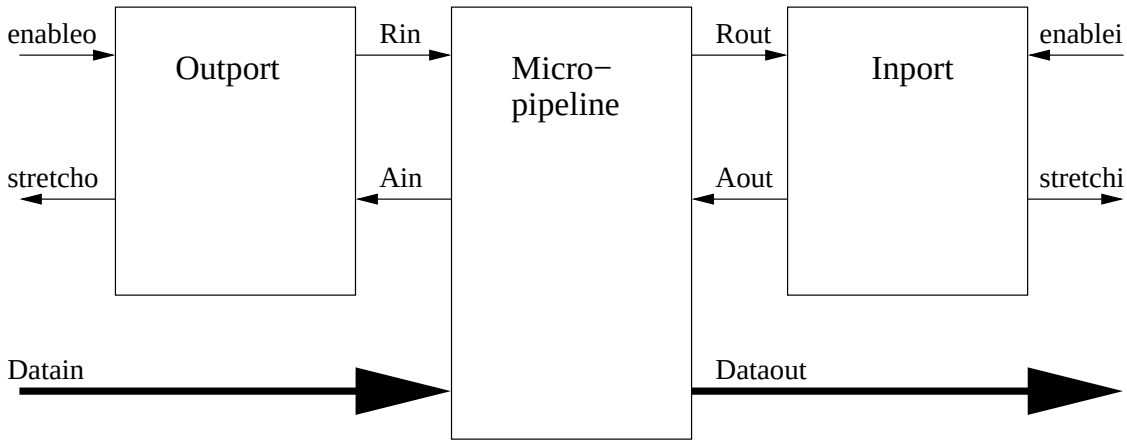


Fig 22. The first circuit simulated.

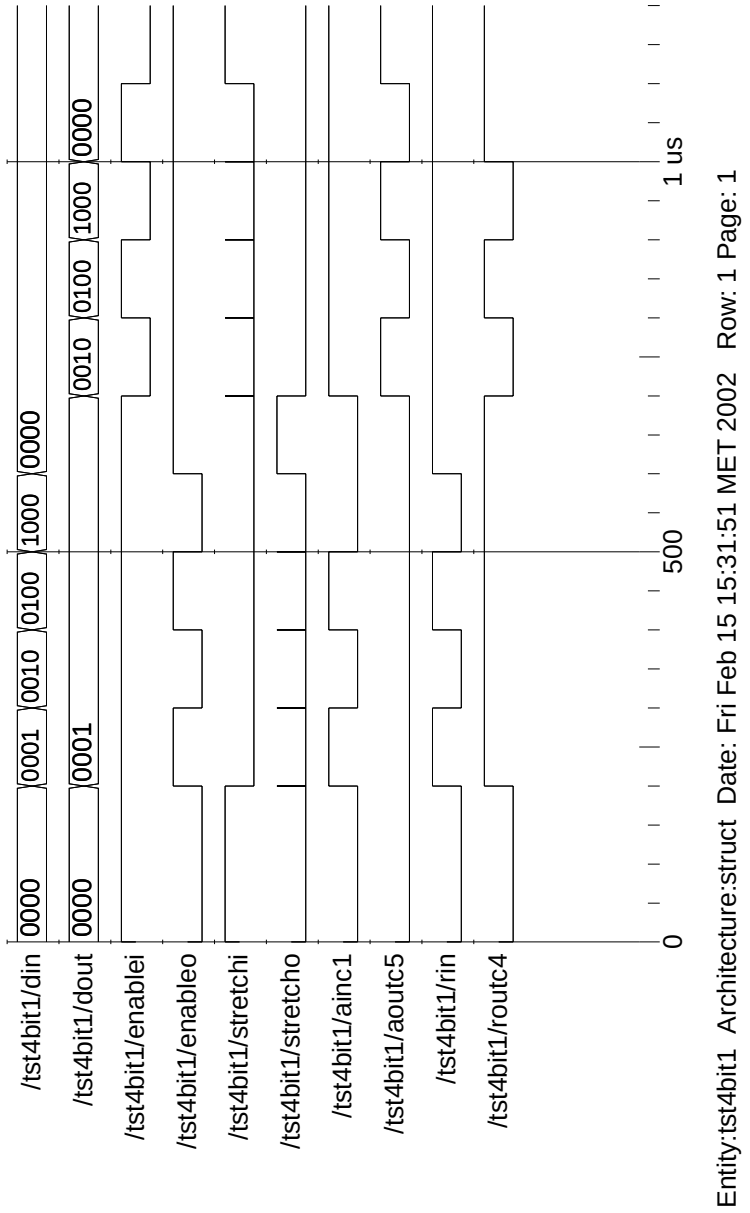


Fig 23. Simulation example from the first circuit.

Then the wrapper was simulated in a context. A four-bit adder was created as shown in fig 24. There were three asynchronous buses used because the adder had two inputs. Thus this was also a test if the wrapper worked with multiple inputs. The circuit was designed to create its own inputs and input them to the adder via two asynchronous buses. The output from the adder was then sent via another bus to the output. The circuit uses three different stretchable clocks.

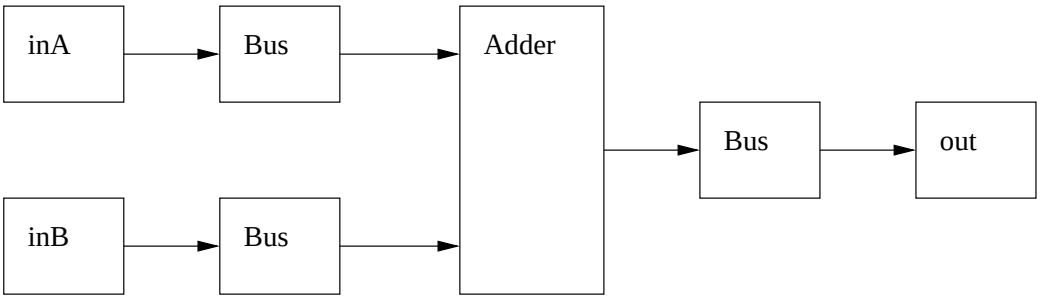


Fig 24. A 4-bit adder with wrappers and FIFO. The Bus consists of an outport, a FIFO and an inport.

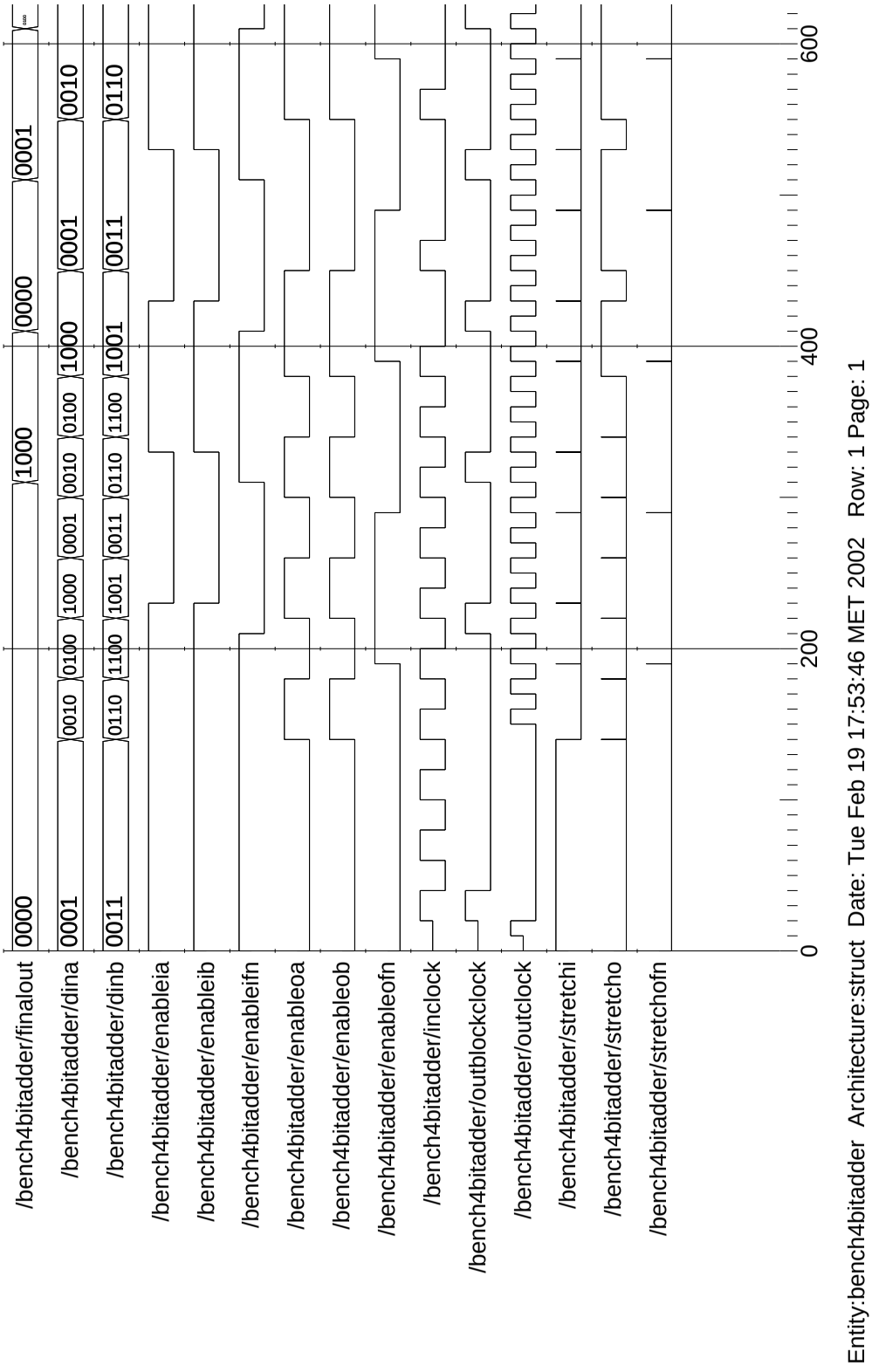


Fig 25. Simultaion of the 4-bit adder with wrappers.

5.4 Synthesis

The synthesis was made for a Xilinx XC4010XL FPGA. The FPGA technology consists of look-up-tables, flip-flops and a connection network.

The synthesis of the VHDL-model proved to be a bit difficult because Leonardo is designed for clocked circuits, but the interesting thing was to see if it was possible to synthesize the design anyway. The first problem was to make a Muller-C element. Using a behavioral description with a process and if-clauses did not work because the synthesis tool always thought that there should be a clock somewhere. The first attempt was to use a SR-latch and some logic as shown in fig 26 but with a “process” latch it did not work. LeonardoSpectrum synthesized it in an unexpected way. It was impossible to make synthesize it without the tool wanting to put a clock somewhere. At some point the reset input on the latch was assumed to be the clock input! The solution was to make the SR-latch in the classic way with two nor gates. This worked, but when the Muller-C element was used in a bigger context Leonardo had a lot of warnings about combinatorial loops. As shown in fig 27 the Muller-C element was synthesized as a look-up table with feedback which hopefully will make it reliable when it is used in a context.

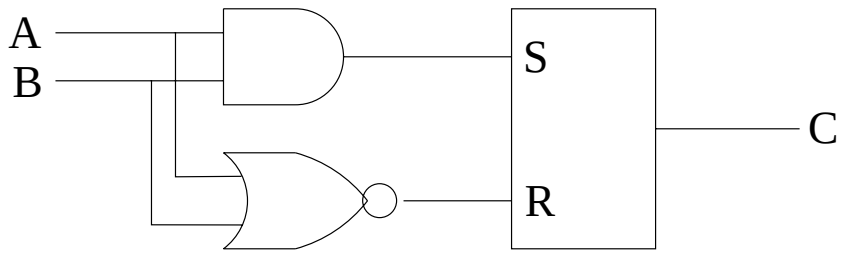


Fig 26. Solution of Muller-C element.

.

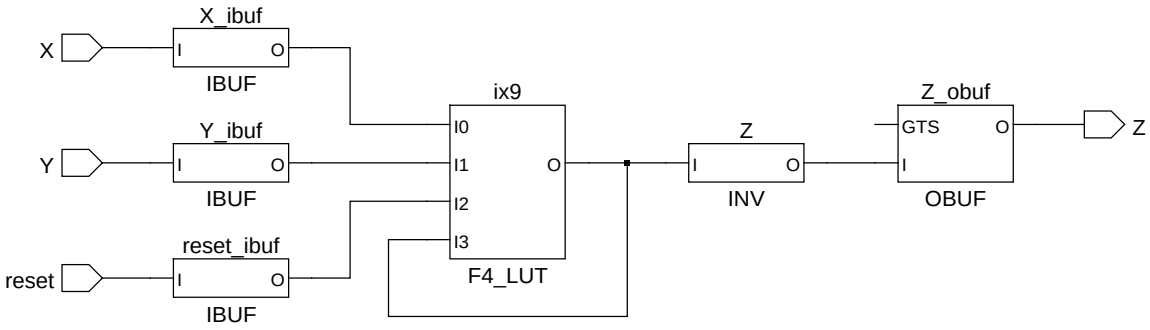


Fig 27. Technology schematic of Muller-C element.

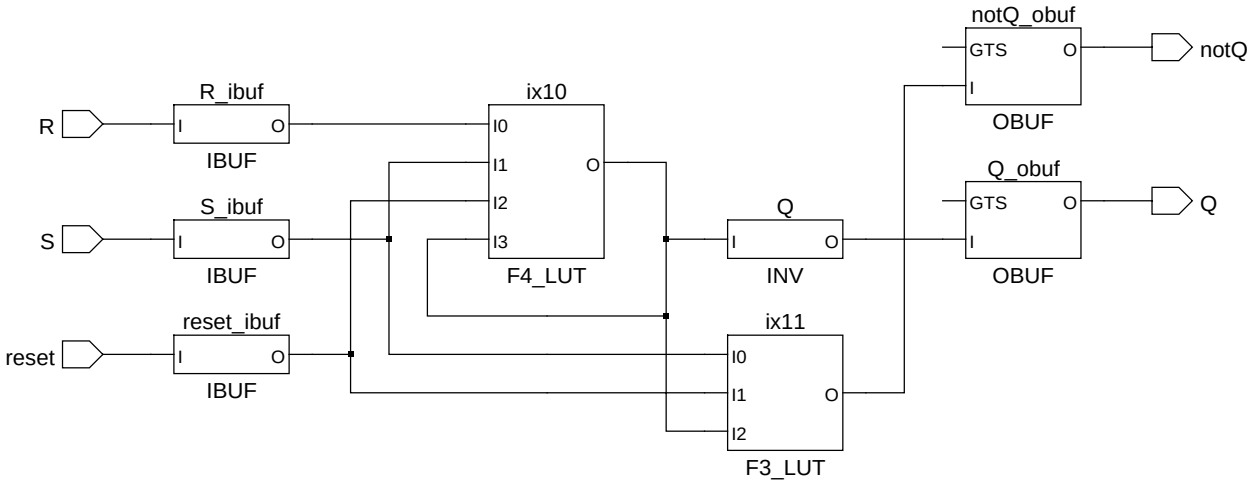


Fig 28. Technology schematic of SR-latch.

In fig 28 is the technology schematic of the SR-latch shown. The interesting thing was that the SR-latch was synthesized in a more complicated way then the whole Muller-C element, which indicates that LeonardoSpectrum did optimize the Muller-C element.

Next to synthesize was the micropipeline which consisting of the Muller-C elements in the control circuitry and switches which were made with if statements. This time the synthesis worked well, the tool seemed to understand that a switch doesn’t need clocking. The inport and outport circuits were also easy to synthesize since they only consisted of combinatorial logic with no feedback.

5.5 Synthesizing the complete circuit

To test the whole design a circuit according to fig 29 was first synthesized.

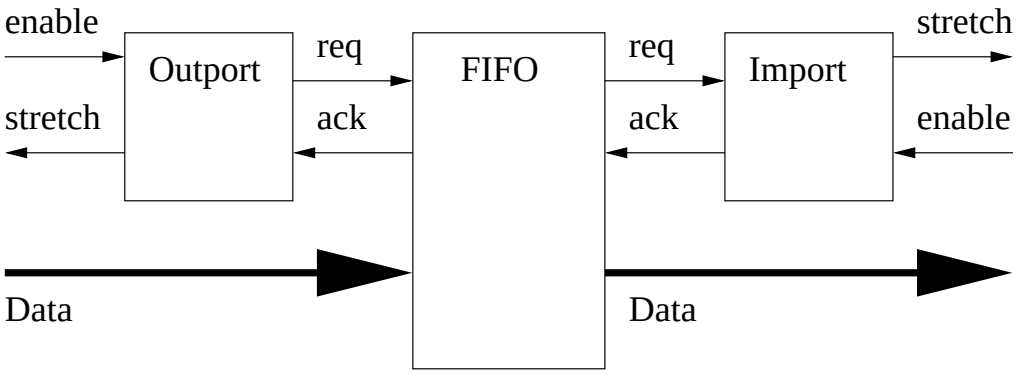


Fig 29. The first circuit synthesized.

This was not easy because for some reason when starting Leonardo from HDL Designer the input files must have been entered in the wrong order. The circuit coming out of the synthesis always had some part that was not synthesized at all! This problem was probably due to the fact that the circuit was partly asynchronous. The solution to this problem was to load the input files manually. Of course this was very strange and the usability of Leonardo for asynchronous synthesis thus seems limited.

A very simple synchronous circuit was also created to test the idea of the stretchable clock, and it also worked after loading the input files manually. The problem was that a ring oscillator for the clock could not be used so an external clock was used instead, and it was set to zero when stretch was high. For more critical applications this could be a problem since it is not a true stretchable clock.

Maybe the biggest problem with using synthesis tools to create asynchronous circuits is how to know if the circuits are hazard free. The circuits worked when tested in the FPGA, but it is difficult to guarantee that hazards never occur. The synthesis tool will optimize the circuits, and then the extra gates needed to prevent hazards will removed.

5.6 Example: Implementation of Muller-C element

The first idea was first simply to make a Muller-C element by using the code shown in fig 30.

```
-- renoir header_start
--
-- VHDL Architecture basic_element.MullerC.interface
--
-- Created:
--          by - olofm.es_exj (delling.isy.liu.se)
--          at - 17:02:22 09/27/01
--
-- Generated by Mentor Graphics' Renoir(TM) 2000.3 (Build 2)
--
-- renoir header_end
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY MullerC IS
  PORT(
    A     : IN     std_logic  ;
    B     : IN     std_logic  ;
    reset : IN     std_logic  ;
    C     : OUT    std_logic
  );

-- Declarations

END MullerC ;

-- renoir interface_end
ARCHITECTURE source OF MullerC IS
BEGIN
 PROCESS(A,B,reset)
 variable oldc : std_logic;
 BEGIN
  if reset = '1' then
   C <= '0';
   oldc := '0';
  elsif A = '1' and B = '1' then
   C <= '1';
   oldc := '1';
```

```
  elsif A = ’0’ and B = ’0’ then
   C <= ’0’;
   oldc := ’0’;
  elsif A = ’1’ and B = ’0’ then
   C <= oldc;
  elsif A = ’0’ and B = ’1’ then
   C <= oldc;
  end if;
 end process;
END source;
```

Fig 30. The first Muller-C element in VHDL.

This idea worked well during simulation, but it did not work well with the synthesis tool. As mentioned above the synthesis tool could not understand that a process without clocking could exist. Then was another solution to the problem tested. The idea was as shown above in fig 26 to connect gates to a SR-latch. First this did not work, but that was because a process was still used to make the SR-latch. The final solution was to do the SR-latch the classic way with two nor gates. In fig 31 the HDL-designer schematic of the final solution is shown, and in fig 32 the VHDL code for the two blocks.

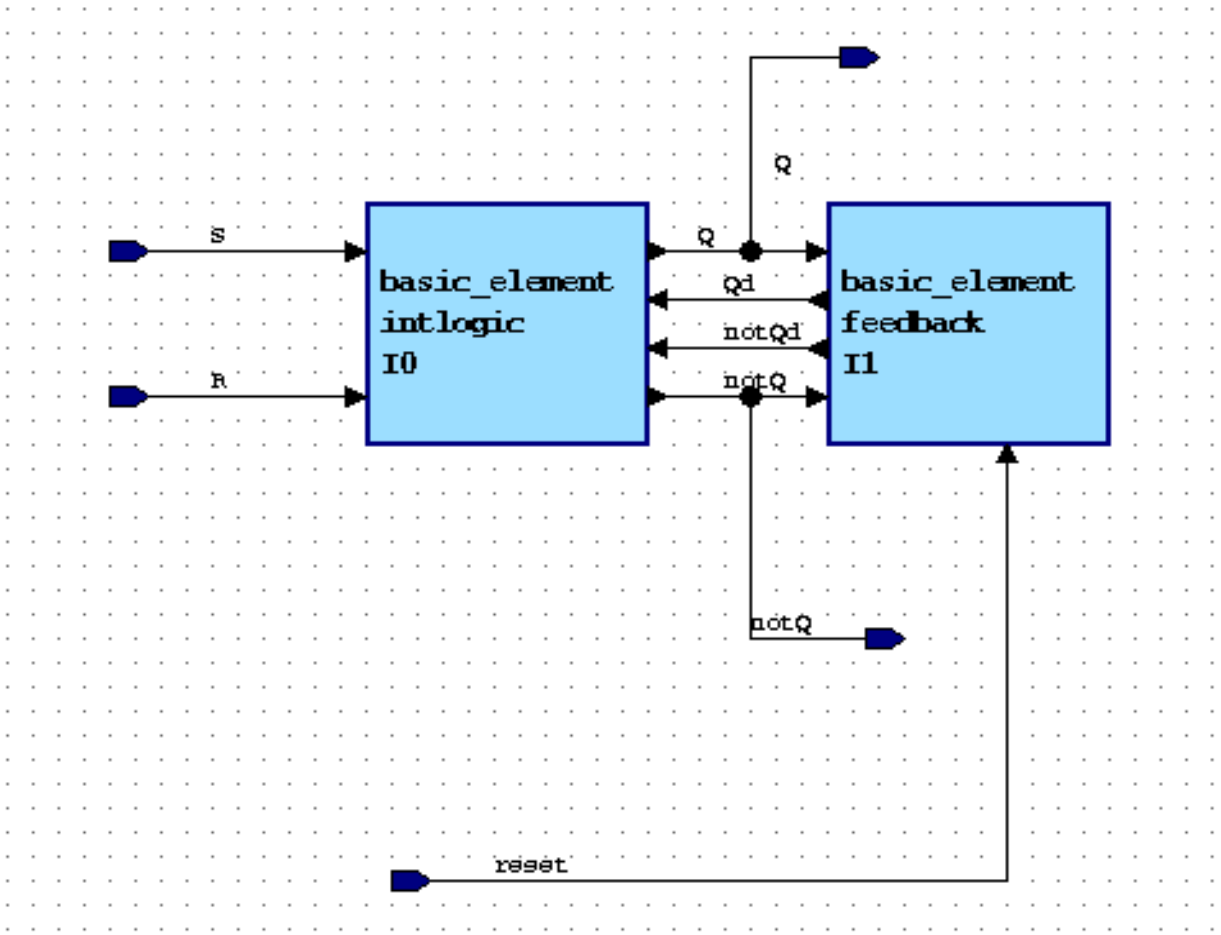


Fig 31. HDL-designer schematic of an SR-latch.

```
ARCHITECTURE source OF intlogic IS
BEGIN
 notQ <= not (S or Qd);
 Q <= not (R or notQd);
END source;

ARCHITECTURE source OF feedback IS
BEGIN
 process(reset,Q,notQ)
 BEGIN
  if reset = ’1’ then
   Qd <= ’0’;
   notQd <= ’1’;
  else
   Qd <= Q;
   notQd <= notQ;
  end if;
 end process;
END source;
```

Fig 32. VHDL-code for the two blocks in the final Muller-C element.

This solution also worked well during synthesis so this was the Muller-C element used in the final version of the VHDL micropipeline.

5.7 Muller-C element in CMOS VLSI

When a Muller-C element should be implemented in VLSI the circuit shown in fig 33 is usually used [9]. It is also possible to make a dynamic Muller-C element, but that seems to be rather dangerous when they are used in asynchronous circuits since it is never known how long a signal must be hold before it is changed.

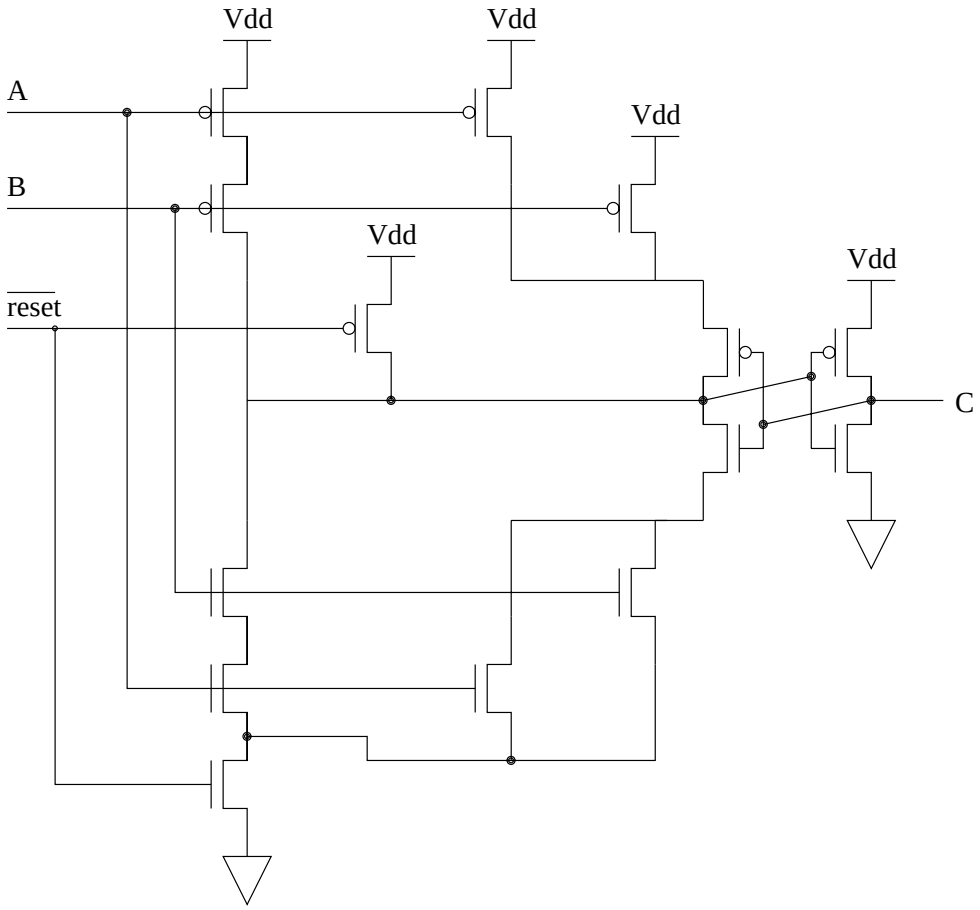


Fig 33. CMOS implementaion of a Muller-C element [9].

It is very important to notice that usually the reset signal is not usually shown on the schematic symbol of Muller-C elements, but as the element in a way works as a memory it is still important to reset the elements when the circuit is started.

6 Conclusions

The concept of GALS circuits are of course very interesting and theoretically they have benefits that seem to make it inevitable that they will become the standard for integrated circuits. This will on the other hand not guarantee that they will be used commercially. Maybe they will just be something that will be forgotten. Fully asynchronous circuits can also claim to have benefits over synchronous circuits, but they have never been able to compete, at least not for the last 30 to 35 years [1]. Of course there is the problem of compatibility. According to [1] an important problem for asynchronous circuits to be used commercially is that they have problems communicating with other clocked parts. For example, if an asynchronous microprocessor is used it might have problems to work with other clocked chips. The solution is of course to use asynchronous wrappers around the clocked parts, but this means that every circuit has to be custom made, at least until the fully asynchronous/GALS paradigm has become standard. The problem with communication between the old and new chips can cause a big problem for the GALS paradigm to become the standard paradigm. As research about fully asynchronous circuits also will make them more easy to design, GALS circuits might be an unnecessary paradigm. GALS circuits may only be a transition phase between synchronous and asynchronous technology.

The other thing that has been tested was the possibility to make asynchronous circuits using VHDL. The problems here were two: First, the circuits must be hazard-free. Since it is hard to know whether LeonardoSpectrum will optimize the hazard-free nets or not, it will be difficult to guarantee that the circuit will work under all circumstances. This makes it unlikely that the synthesis will end with a working circuit. In the wrapper design the logical nets were already the simplest possible, which might be why they worked anyway. The second problem is that the solution to the problem that LeonardoSpectrum did not finish the synthesis in some cases was to reload the input files in a new order, and the order was found out by chance. This was possibly because the project was rather small, but of course this is not possible to do when the circuits grows bigger. So the conclusion is that it should be possible to make asynchronous or GALS circuits using VHDL, but LeonardoSpectrum should not be used. There does not exist any commercial synthesis program for asynchronous design yet, but it should be possible to write one.

References

[1]Myers C. J., ’Asynchronous Circuit Design’, Wiley 2001

[2] Danielsson P.-E., Bengtsson L., ‘Digital teknik’, Studentlitteratur 1996

[3] Yun K. Y., Dill D. L., ‘Automatic Synthesis of Extended Burst-Mode Circuits’, IEEE Transactions on Computer-Aided design of integrated circuits and systems, Vol. 18 no 2, February 1999

[4] Sutherland I. E., ‘Micropipelines’, Communications of the ACM, June 1989 Volume 32 Number 6

[5] Liljeberg P., Plosila J., Isoaho J., ‘Synchronous/Asynchrnonous Interface Unit for IP based SoC systems’, NORCHIP 2000

[6] Bormann D. S., Cheung P. Y. K., ‘Asynchronous Wrapper for Heterogenous Systems’ In Proc International Conf. Computer Design (ICCD), Oct 1997

[7] Muttersbach J., Villiger T., Fichtner W., ‘Practical Design of Globally-Asynchronous Locally-Synchronous Systems’, Advanced Research in Asynchronous Circuits and Systems, 2000 (ASYNC 2000) Proceedings. Sixth International Symposium on, 2000

[8] Fuhrer M. R. et. al., ‘MINIMALIST: An Environment for the Synthesis, Verification and Testability of Burst-Mode Asynchronous Machines’, Columbia University Computer Science Dept. Tech Report #CUCS-020-99

[9] Lu S. L., ‘Improved design of CMOS multiple-input Muller-C-elements’, Electronic letters 16th September 1993 Vol. 29 No. 19

LINKÖPING UNIVERSITY
ELECTRONIC PRESS

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Olof Manbo