NTNU – Trondheim
Norwegian University of
Science and Technology

Synthesis of low-energy near-threshold SHMAC processor core

Miguel Zapatero

Embedded Computing Systems
Submission date: June 2015
Supervisor: Snorre Aunet, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

NTNU
Norwegian University of Science and Technology

Synthesis of low-energy near-threshold SHMAC processor core

Miguel Zapatero Rodríguez

June 2015

Supervisor: Professor Snorre Aunet
Department of Electronics and Telecommunications
Norwegian University of Science and Technology

Project assignment

Student’s name: Miguel Zapatero Rodriguez

Course: TFE4915 - Circuits- and Systems Design, Master's Thesis.

Project title: Synthesis of low-energy near-threshold SHMAC processor core.

Project description:

- Improve a basic cell library that includes: inverter gate, NAND gate, NOR gate and a D Flip-flop creating the layout and optimizing it to work with nearthreshold voltage supplies.
- Use this library to synthesize a core.
- Simulate this core studying the possible improvements in power consumption and how the performance is affected.

Abstract

In this project an AMBER processor's ALU is synthesized to operate in the near/subthreshold region. The main motivation for this work is to study the possibility of implementing near/subthreshold modules in the SHMAC processor. These modules may be used for low requirement operations and are supposed to decrease the power consumption of the system.

Near/subthreshold operation is a developing method to reduce power consumption in static MOS technology. It consists on using near/subthreshold voltages to reduce the dynamic power of the system. The dynamic power consumption is proportional to the square of the operational voltage so, in theory, reducing the supply voltage can significantly reduce the power usage of the system.

A cell library consisting in NAND,NOR, XOR, inverter and D Flip-flop gates is created aimed to work with a near/subthreshold voltage supply. This was done using CADENCE and standard 65nm CMOS technology.

AMBER processor's ALU is synthesized using the custom library created, and simulated for 400mV.

Preface

This report is written as a master thesis for the Master degree program at the University of Science and Technology (NTNU) of Trondheim. It is written during the spring 2015. The thesis is a continuation of the specialization project of the same name.

The supervisor, Professor Snorre Aunet, proposed this topic but the objectives and approach were chosen by me.

This master thesis has proven a challenge due to the difficulties of using new complex programs such as Cadence but it has given me knowledge about cmos layout and synthesis process.

I would like to thank my supervisor, professor Snorre Aunet, from the support given during the project.

I would also like to thank PHD student Benjamin Bjørnseth and Master student Henrik Fegran for the help provided during in the synthesis process.

Contents

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The introduction consists of a brief description of the background and motivation for this project, and a brief explanation of the approach followed.

1.1 Background and motivation

Nowadays the dark silicon effect is a main concern in the development of computing systems. The escalation in the transistor sizes has not been followed by a similar reduction in power consumption and this has led to electronic systems in which only a subset of the resources can be power at the same time.

Heterogeneous computer systems are systems that use more than one type of processor to handle different task. The use of specialized processor design for a particular task can increase the performance of the system or reduce its power consumption. Using different modules with different performances and power consumptions is thought to be one of the best approaches in order to overcome this dark silicon limitation.

The EECS department of the NTNU is currently developing a heterogeneous processor (the Single-ISA Heterogeneous many-core Computer or SHMAC) to research the challenges of heterogeneous computer systems. The aim is to exploit the heterogeneous processor in an attempt to mitigate the dark silicon effect.

On the other hand, reducing the power consumption is a main research topic in the electronic field and near/subthreshold supply is one of the main trends in this research.

The power consumption in static CMOS technology is caused by three contributions; these are short circuit, leakage and switching power consumption. The switching power is proportional to the square of the voltage supply and the leakage and short circuit power are also proportional to the voltage supply so reducing it can lead to an important reduction in power consumption. The problem is that the performance of the circuit also reduces with the reduction in voltage supply so it is important to study the effect of this reduction and try to achieve the best trade-off between power consumption and performance.

Using a near/subthreshold module in the SHMAC platform will allow the ECCS to investigate the possibility of using a very low power consumption unit for computations with low speed requirement.

1.2 Approach

This project is a continuation of the specialization project of the same name. In this specialization project due to the impossibility of synthesising in near/subthreshold voltages with the provided library, a small cell library was made (based on schematic design).

For this master thesis the first step is to do an extensive research on near/subthreshold MOS technology. It is important to acquire this knowledge to be able to understand the possibilities and limitations of this technology and, later on, to study the characteristics of the near/subthreshold implementation.

The second step consists in improving the previous library by creating the layout of the cells and extracting its parasitics. This implies familiarizing with the tool Virtuoso of Cadence.

The third step will be compare the behaviour of the cells and see how the parasitics affect the functionality.

The fourth step will be to synthesize the ALU of the AMBER processor which is given in a hardware description language (Verilog). This requires familiarizing with the use of different programs: encounter RTL compiler, Synopsis and Virtuoso.

The fifth step is simulating the synthesized ALU to study how the reduction in the power supply influence in the power consumption and performance which respect to the normal voltage supply.

2 Theoretical Background

2.1 Dark Silicon

In accordance with Moore's law, the number of transistors that can be implemented per unit of area doubles every two years (see Figure1). Unfortunate, the power consumption has not decreased in the same proportion. This has led to electronic systems consuming more and more energy and increasing their power density (see Figure2).

This increase in the energy consumption has three main adverse effects:

- Economic impact: For large systems (such as data centres) the cost of the energy (in the form of electricity) is one of the main concerns.
- Battery life: Portable electronic devices are more and more powerful but the batteries technology is not able to follow the increase in power consumption. This leads to lower battery live.
- Thermal issues: Electronic devices with a high grade of integration are nowadays not able to work at full performance due to the impossibility of dissipating all the heat that the power dissipation creates. This make impossible to power all the devices in a system at maximum frequency.

The third point leads to the so call "Dark Silicon effect" that consist in the impossibility of power all the transistors in a chip at the same time.

To deal with this Dark Silicon effect, several approaches are been researched; been heterogeneous systems one of the most promising.

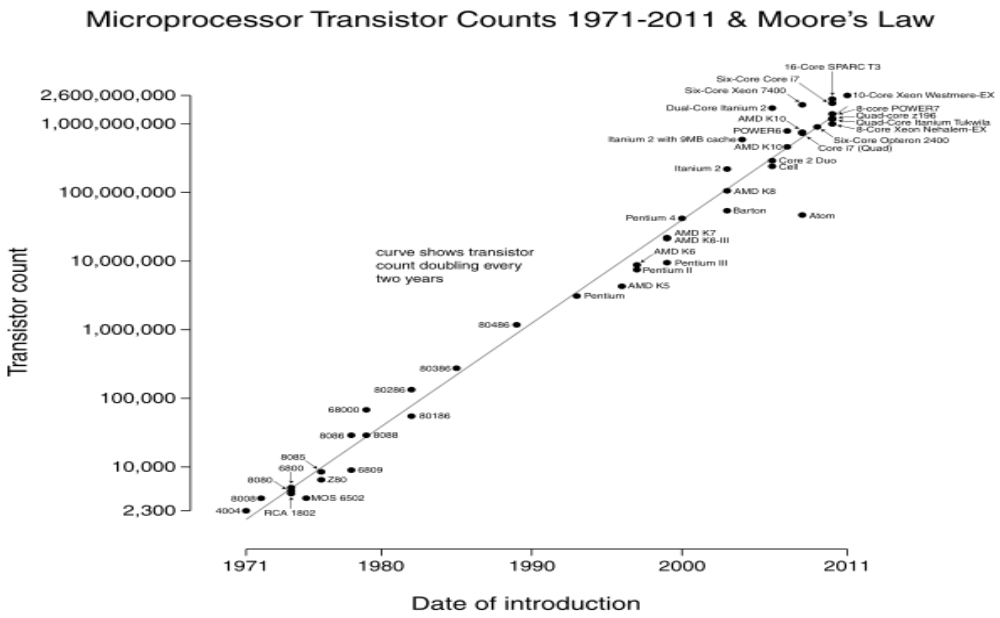
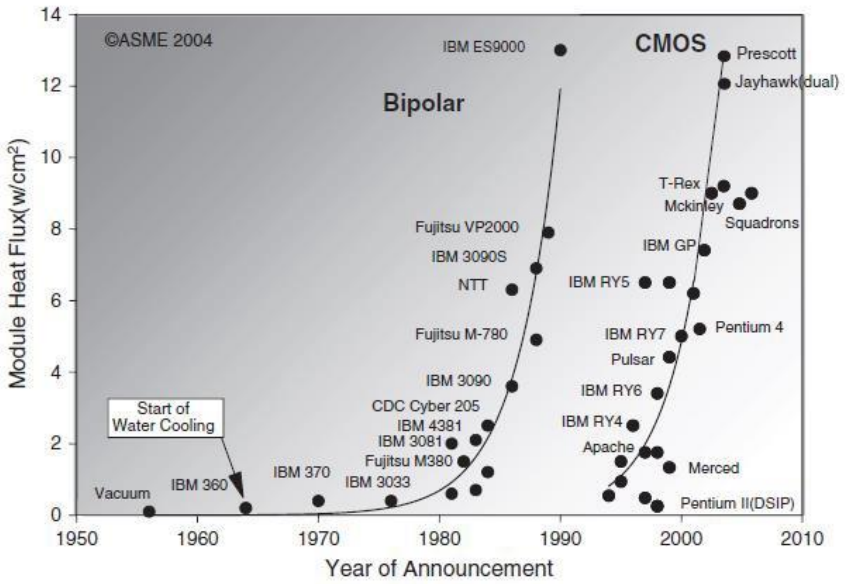


Figure 1. Moore's law



Introduction of CMOS over bipolar bought the industry 10 years (example: IBM mainframe processors)

[Ref: R. Chu, JEP'04]

Figure 2, power density

2.2 CMOS Logic

CMOS technology, or complementary metal oxide semiconductor, is a widely used technology due to the low static consumption. Since one transistor of the pair is always off, the series combination draws significant power only momentarily during switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic.

CMOS technology also has the advantage of been extremely easy to use to create logic circuits given its logic function.

CMOS digital gates are made from analog nMOS and pMOS transistors. This means that the behaviour of the gates will have an undesired analog component despite the design. Some of this effects are raise time, fall time and leakage current.

During the creation of the gates, several parameters can be modified to try to minimize these effects being the most important the width and the length in the nMOS and pMOS transistors.

2.3 Power consumption in CMOS circuits

The increase of power consumption and the appearance of the dark silicon effect have led to an increase in the research of low power consumption electronic devices.

The power consumption in CMOS circuit has three causes: dynamic, static and short circuit power consumption. The total power consumption can be express as the addition of these three powers.

$$Ptotal = Pdynamic + Pstatic + Pshort\ circuit$$

Technology improvements led to decreased C_L, I_{sc} and $I_{leakage}$ that help decreasing the power consumption and architecture design can decrease the activity reducing the power consumption even further. The problem is that the increase in the number of transistors may cause these improvements to be insufficient.

The next approach is reducing V_{DD} but it does not come without a setback. Reducing V_{DD} also reduces the speed at which the circuit can operate. This is because with lower V_{DD} the currents are smaller and it takes longer to charge the capacitances (both internal and in the load). When the voltage supply is lowered to a point where it is close or even lower than the threshold voltage we said that is operating in the near/subthreshold region.

In traditional CMOS circuits the main cause of power dissipation was the dynamic power consumption or switching power, but the appearance of newer technologies in which the leakage current increases and the dynamic power decreases has made the static power consumption an important factor to take into account. This is especially true for circuits working in the near or subthreshold region due to the increase in the delay.

2.3.1 Dynamic or Switching power

The dynamic power consumption or switching power is causes by the activity of charging and discharging the load capacitances when the circuits state changes.

$$Switching\ power = \alpha \times f_{clk} \times C_L \times V_{DD}^2$$

Where:

- α is the activity rate of the circuit.
- f_{clk} is the frequency at which the circuit is working.
- C_L is the load capacitance.
- V_{DD} is the voltage supply.

This source of power consumption can be easily decrease by decreasing V_{DD} and that is the objective of this thesis.

2.3.2 Short circuit power

The short circuit power is due to the brief period in which the NMOS and PMOS transistors in the CMOS circuits are conducting at the same time when a logic changes occur. During this time there is an open path between power supply and ground.

$$Short\ circuit\ power = I_{sc} \times V_{DD}$$

Where:

- I_{sc} is the short circuit current.
- V_{DD} is the voltage supply.

This source of power consumption is the least important one and will not have a big impact in the circuit power consumption.

2.3.3 Static or leakage power consumption

The static or leakage power consumption is due to the leakage current that exist in the transistors. In complementary MOS technology, half of the transistors will have leakage between drain and source leading to some power consumption.

$$Leakage\ power = I_{leakage} \times V_{DD}$$

Where:

- V_{DD} is the voltage supply.
- $I_{leakage}$ is the leakage current.

As the dynamic power decreases the leakage power takes more and more importance, been significant when working in the near/subthreshold region.

2.4 Near/subthreshold region

As the voltage supply is lowered the circuit power consumption decreases and so does the performance of the circuit. Due the increase in the delay of the operations the leakage power increase its importance (the operations takes longer so the transistors are leaking for more time) and, in some points in the subthreshold region, it may even be bigger than the switching power (see Figure 4).

To measure the performance of a circuit we make use of two variables: the energy consumption per operation (related to the power consumption and the time it takes to realize an operation) and the frequency (the maximum speed the circuit can achieve).

In the nearthreshold region, reducing the supply voltage led to a reduction in the energy per operation and frequency that are of the same order (see figure 3).

In the subthreshold region, reducing the supply voltage decrease the energy per operation but the frequency may decrease in a much bigger amount. If the supply voltage decrease too much, the gain obtain by the lowered switching power is counteracted by the increase in the leakage power(see figure 3).

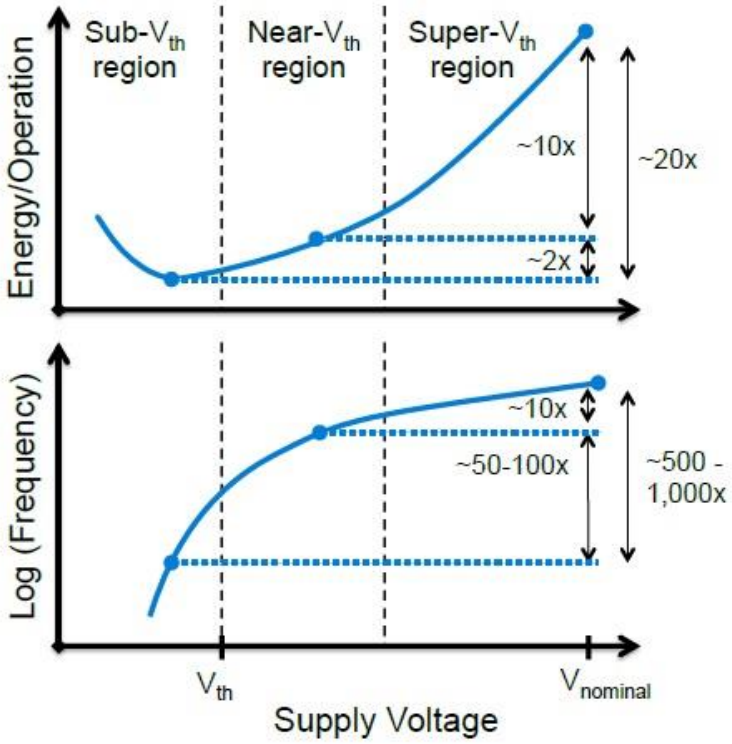


Figure 3, energy per operation and frequency

When using near/subthreshold technology, it is important to study the speed and power consumption requirements of the system to be able to achieve the maximum energy savings with the lowest speed diminish possible.

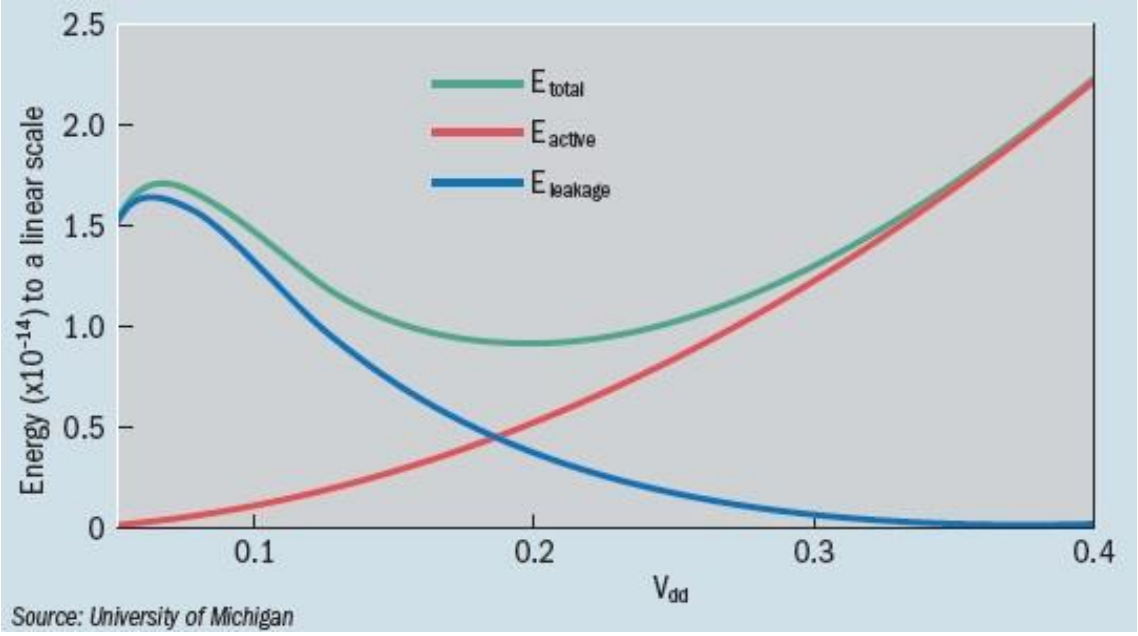


Figure 4, energy/V_{dd}

Near/Subthreshold technology presents other problems. The most important of this are:

- Increased performance variation.
- Increased functional failure.

When working in near/subthreshold region small changes in the transistors can cause an important variation in the performance of the system and the possibility of a critical functional failure also increase. These problems become more and more important as the voltage supply is reduced (see figures 5 and 6).

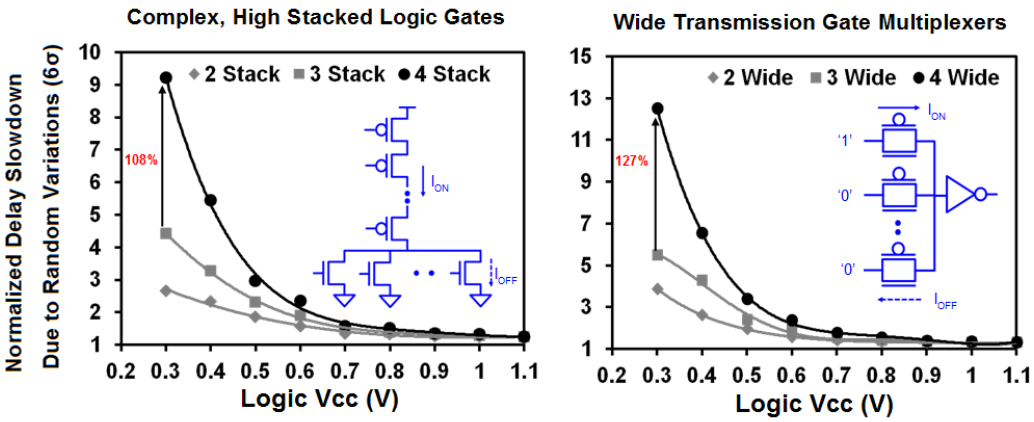


Figure 5, variation in performance

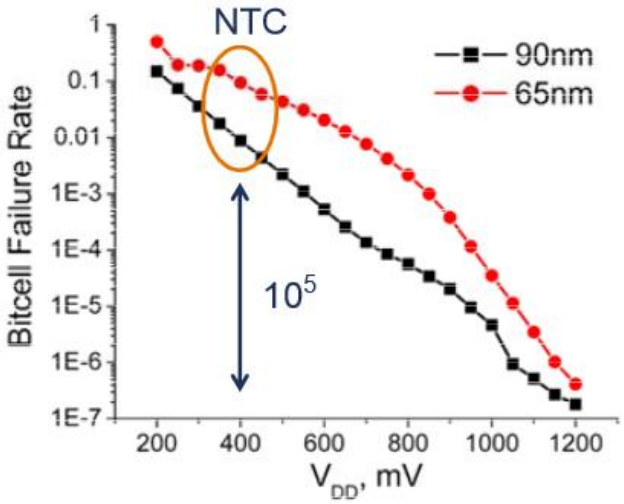


Figure 6, failure rate

To reduce the impact of the random variations it is important to reduce the number of transistors stacked in a gate (see Figure 5). This can be easily achieved by using only 2 or 3 input gates. The downside of this is a reduction in the optimization in the synthesis process. There are other techniques like body biasing (adjusting the threshold value for the transistors) but they are usually too expensive to be used in most systems.

Reducing the probability of failure in a circuit working in near/subthreshold voltage is difficult and expensive. The only method to ensure that there will not be many failures is used a reduced number of transistors, due to this, subthreshold voltage supply is mainly used in not very complex systems like watches or pacemakers.

2.5 Cell library

A cell library is a set of low level cells that can be used to synthesize larger designs.

The cells are created using standard 65nm PMOS and NMOS transistors. To create the D flip-flop the NAND and inverter gates are used.

The cells have to be able to work in near/subthreshold operation point and need to be adjusted to perform as well as possible. For this, the sizes of the transistors were modified, increasing the width of the transistors when needed. This process used feedback from the simulations

In a library is important:

-To be able to have high density. This can be achieve with a high variety of cells that allow the synthesis tool to optimize the design and with cells as regular as possible.

-To have as few parasitics as possible. This can be achieve during the layout design.

The cell library done during the specialization project consists in:

- NAND gate. The truth table is shown in Table 1.

Table 1, NAND truth table

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

- NOR gate. The truth table is shown in Table 2.

Table 2, NOR truth table

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

- NOT gate or inverter. The truth table is shown in Table 3.

Table 3,NOT truth table

NOT gate	
A	$\overline{A}$
0	1
1	0

- D-Latch. The truth table is shown in Table 4.

Table 4, D-Latch truth table

D-Latch			
E	D	Q_{t+}	MODE
0	0	Q_t	HOLD
0	1	Q_t	HOLD
1	0	0	RESET
1	1	1	SET

These cells only consist on schematics and they need to be improve creating the layout and extracting their parasitics. During the layout creation they will also be optimize within possible with the objective of having a low power consumption and a high frequency of operation.

These cells are enough to synthesize any logic circuit but for optimization purposes another cell was created.

- XOR gate. The truth table is shown in Table 5.

Table 5, XOR truth table

2 Input XOR gate		
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

2.6 Layout

The logical gates do not show the true behaviour of a cell, they are only good for the logical simulation of the cell. To get the real behaviour of a cell the physical representation of the cell has to be taken into account. This is the layout level and is the lowest level of representation of a cell.

The layout is a representation of the silicon with the different layers of metal polysilicon and dopants. It also includes the contacts and vias where the cell will be connected.

To create the layout of a cell there are several rules that have to be follow. This rules included:

- Distance between layers. The minimum distance at which different layersc an be placed.
- Width areas. Minimum width that lines or areas can have.
- Contacts . The contacts have to be along the edge of the chip.
- Wells. Wells that have different potential have to be separated a minimum distance between themselves.

2.6.1 DRC

DRC or design rule check. Once the layout of the cell has been created we need to run the DRC. It checks that the layout follows all the design rules and shows you all the infringements that have occurred allowing easier corrections.

This is one of the most tedious steps, until you are familiarized with the rules you will commit many infringements and modifying the layout to correct them is pretty time consuming.

2.6.2 LVS

LVS or layout versus schematics. Once the DRC test is passed is time to place the vias. The LVS checks that the schematic model corresponds with the layout and that all the components are rightly connected. It also checks that no via or path is missing in the layout with respect the schematic.

2.6.3 PEX

PEX or parasitics extraction. At this point the layout is already created. The PEX extracts the parasitics of the design and integrate them into the cell. There are several parameters that can be adjusted to be more or less precise in the extraction, see Figures 7 and 8. Once that all the parameters are chosen the parasitics are extracted, see Figure 9; and the extracted view is shown, see Figure 10.

After the PEX is done a PEX file is created. This file will be used for the simulation.

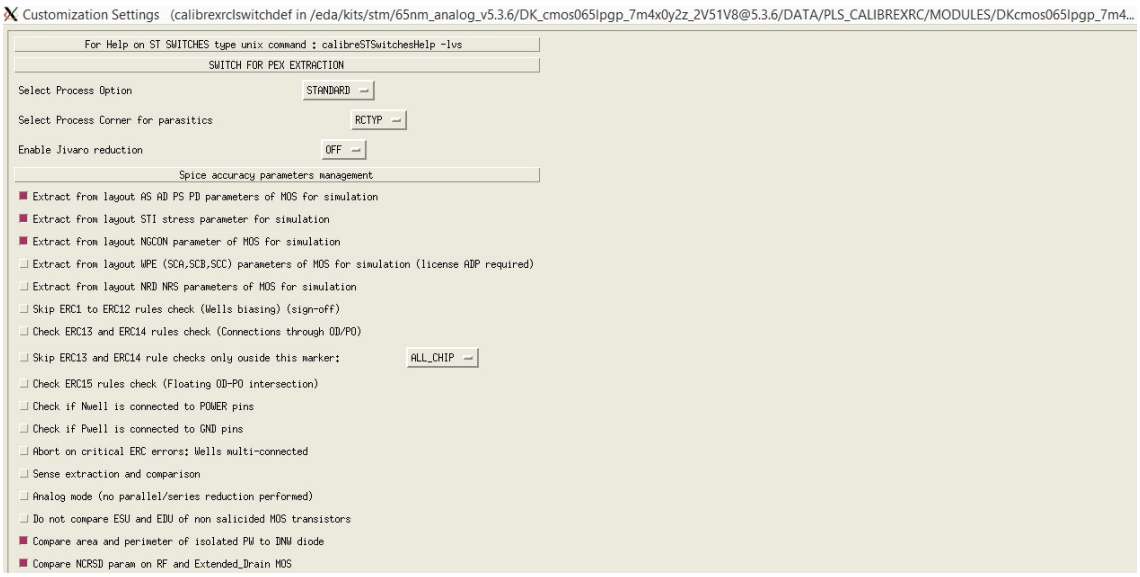


Figure 7, PEX parameters 1

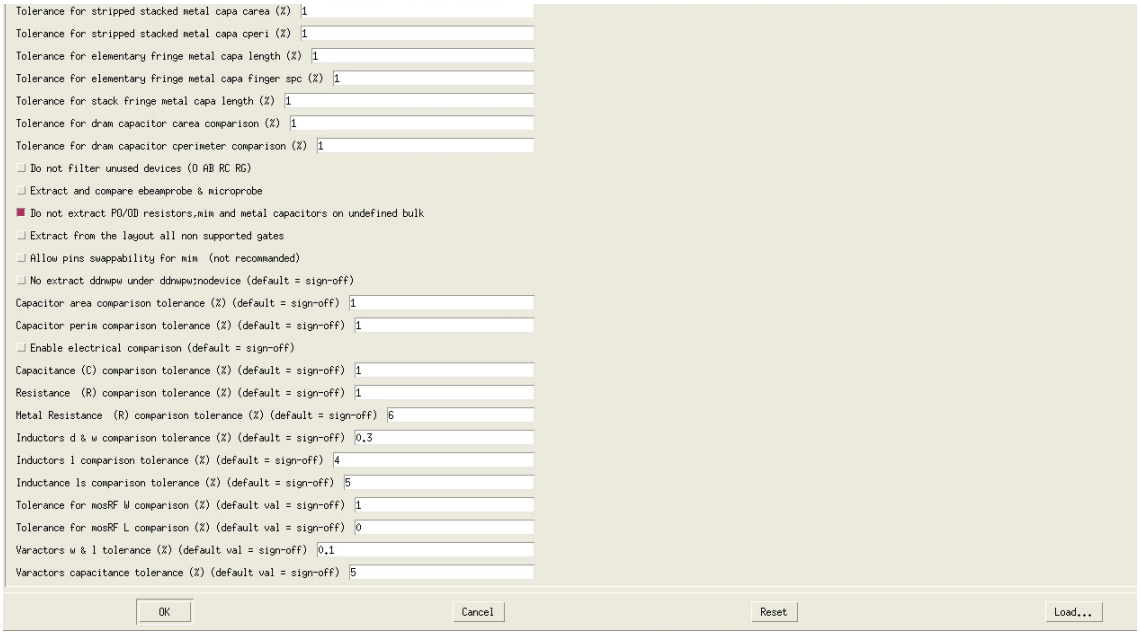


Figure 8, PEX parameters 2

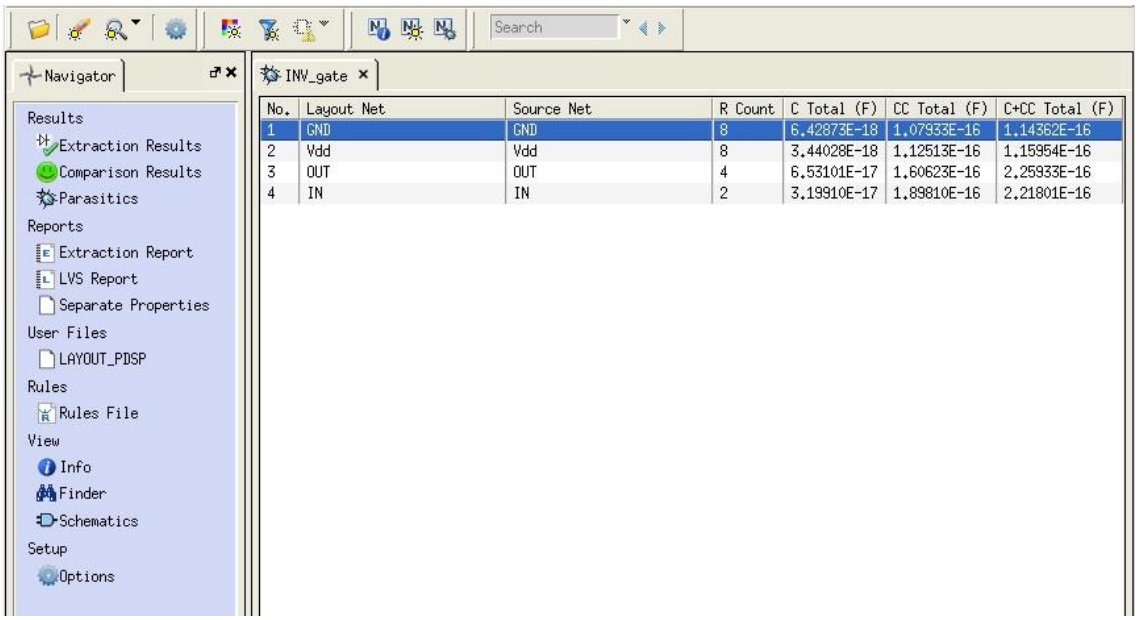


Figure 9, parasitics extraction

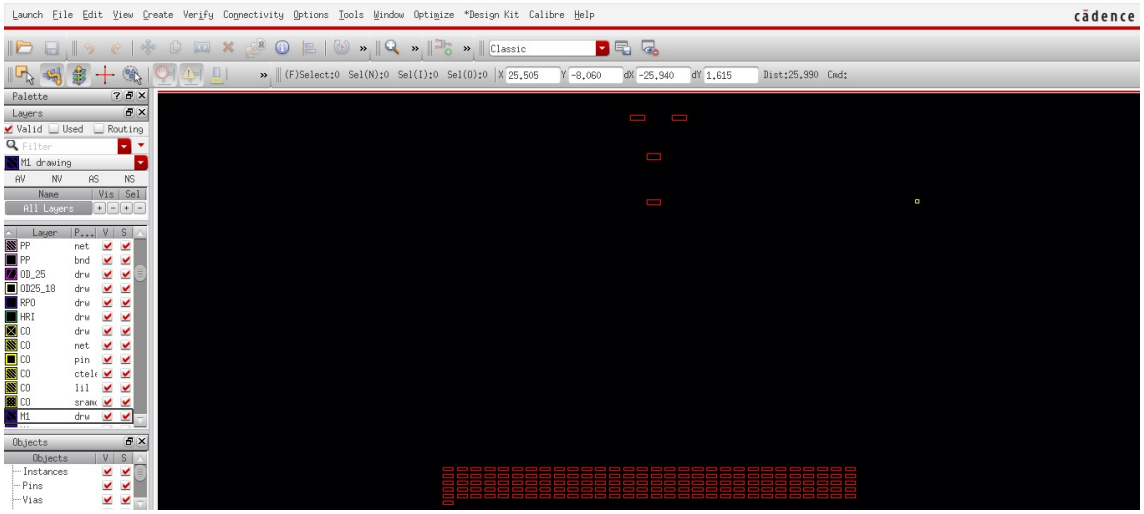


Figure 10, extracted view

2.6.4 Library extraction

After all the cells have pass all the test and their parasitics have been extracted, three different files are needed.

- Abstract view. Obtained by the abstract generator.
- .LEF files. These are library exchange format files . Also obtained by the abstract generator.
- .lib files contain information about the timing and functionality of the cells. They are obtained using Liberate.

2.7 Synthesis theory

The objective of the synthesis process is to get a circuit design (in which we can do power and timing simulation) from an abstract high level model (the given Verilog code for Aurora's ALU). The synthesis process consists in different phases, see Figure 11.

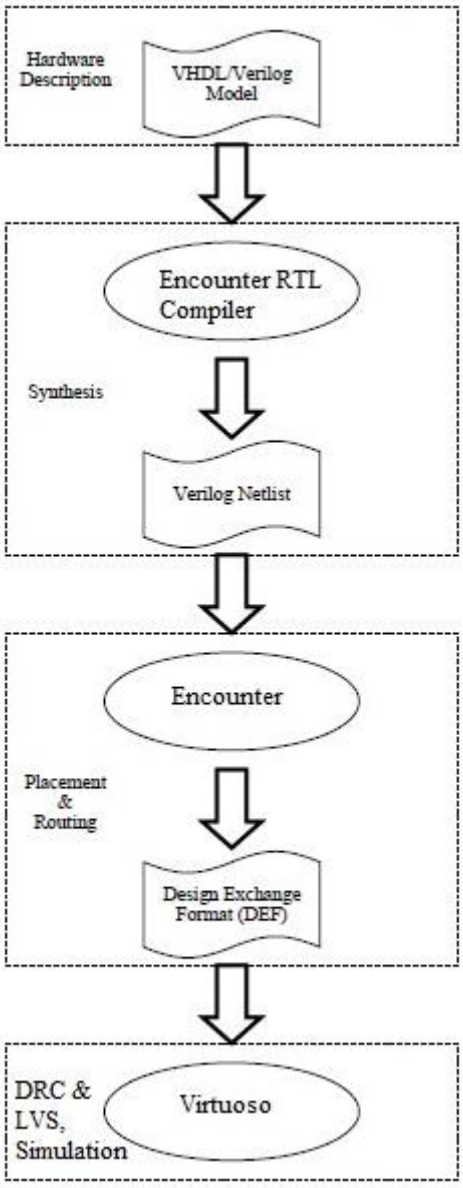


Figure 11, synthesis process

2.7.1 Verilog and VHDL

The design to be synthesized is a 32 bits ALU. This ALU is part of the Amber open source project. This ALU supports the functions: 32-bit add and subtract, AND, OR, XOR, NOT, Zero extent 8-bit numbers.

The ALU is given in Verilog language. Verilog is a hardware description language used to model and simulate hardware designs at different level of abstraction. The ALU is given in a behavioural level of abstraction. See Appendix 2.a.

2.7.2 Encounter RTL compiler.

Given a Verilog behavioural description and a cell library, Encounter RTL compiler transform it into an RTL description of the circuit (written in Verilog), part of the code is shown in Apendix 2.b. This is a list of the pins, wires, components and the way they are connected between them. It is already possible to import this file to Virtuoso for verification purposes, see Apendix 2.c.

2.7.3 Synthesis, place and route

Once the RTL code is ready it is possible to synthesize it with the LEF file from the library. The physical properties contained in the LEF file are used in the synthesiser for the routing. The synthesis can be adjusted for different levels of optimization for a better or worse level of integration (more or less chip surface unused). This is a heavy process that takes a long time. Due to the limitation of using a common server for the synthesis instead of a dedicated computer, this process took a really long time and a good optimization was not possible.

This process results in a .DEF file (Design exchange format) which can be imported into Virtuoso.

Now The DRC and LVS test has to been pass again to make sure that the synthesized circuit follows all the design rules and the layout and schematics correspond. Now it is possible to run a simulation using ERC.

3 Custom design library

The library is based in the cell library done during the specialization project. It contained the inverter, NAND, NOR and D-flipflop gates and later on the XOR gate was included. The behaviour of the gates can be seen in Appendix 1.

The main data to be obtained is:

- Rise time. Time that the signal needs to get from 10% to 90% of its final value.
- Fall time. Time that the signal needs to get from 90% to 10% of its final value.
- Average power consumption. Calculated assuming that the all the values of the inputs have the same probability.

With this data other parameters have been calculated:

- Delay. Taking into account the worst case of the rise and fall time.
- Energy per operation. Using the average power consumption and the delay.

The cells have been optimized within possible by changing the width of the transistors. The main reason to do this is increase the current that is used to charge the capacitances in the cases with worst delay. See Appendix 4.a for the optimization flow.

3.1 Inverter

The inverter is the simplest gate that can be done (see figure 12) but it is important to adjust the sizes of the transistors to get a good characteristic curve (see figure 13).

To get a good curve the width of the PMOS was increased.

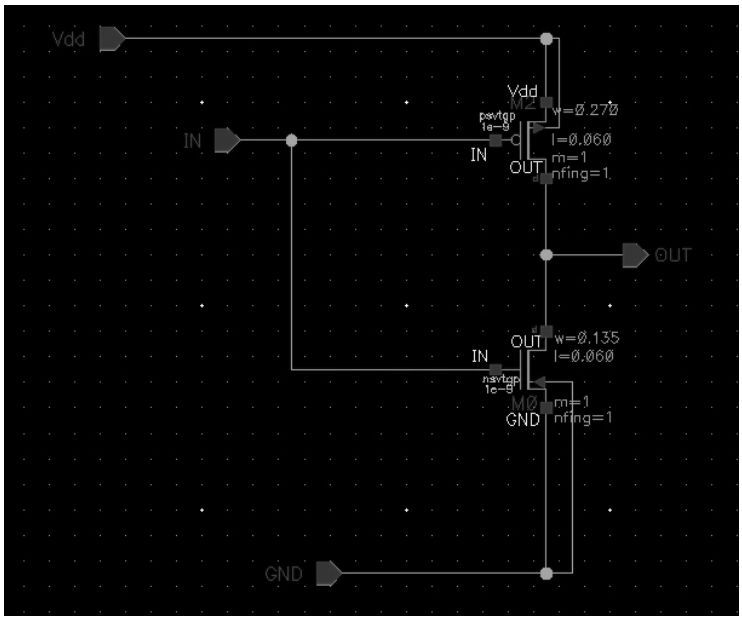


Figure 12, inverter schematics

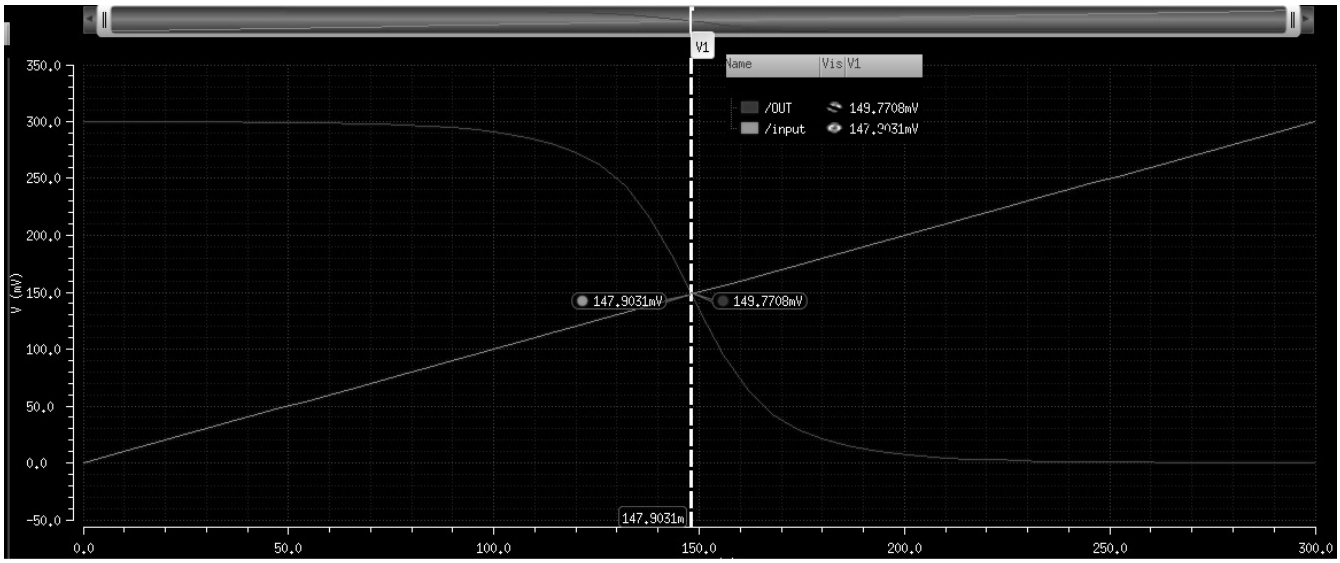


Figure 13, inverter curve

The data obtain in the inverter simulation is shown in table 1. The delay and energy per operation for the different voltages is shown in figure 12.

The worst delay in the inverter happened when the signal had to change from 0 to 1. In this case the delay was conditioned by the rise time.

Table 6, inverter without parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average power	Δpower	Energy/operation	ΔEnergy/operation
0,2	4,28E-10	3,25E-10	6,42E-10	564,9%	7,75E-10	0,9127%	4,9792E-19	5,1552%
0,3	1,29E-10	1,09E-10	1,94E-10	170,3%	1,49E-09	1,7502%	2,8796E-19	2,9813%
0,4	1,12E-10	9,87E-11	1,69E-10	148,3%	2,48E-09	2,9167%	4,1779E-19	4,3255%
0,5	9,31E-11	8,24E-11	1,40E-10	122,9%	4,27E-09	5,0200%	5,9580E-19	6,1685%
0,6	7,82E-11	7,00E-11	1,17E-10	103,1%	8,06E-09	9,4856%	9,4472E-19	9,7810%
0,7	7,03E-11	6,53E-11	1,05E-10	92,7%	1,57E-08	18,4675%	1,6538E-18	17,1225%
1	7,58E-11	7,33E-11	1,14E-10		8,50E-08		9,6587E-18	

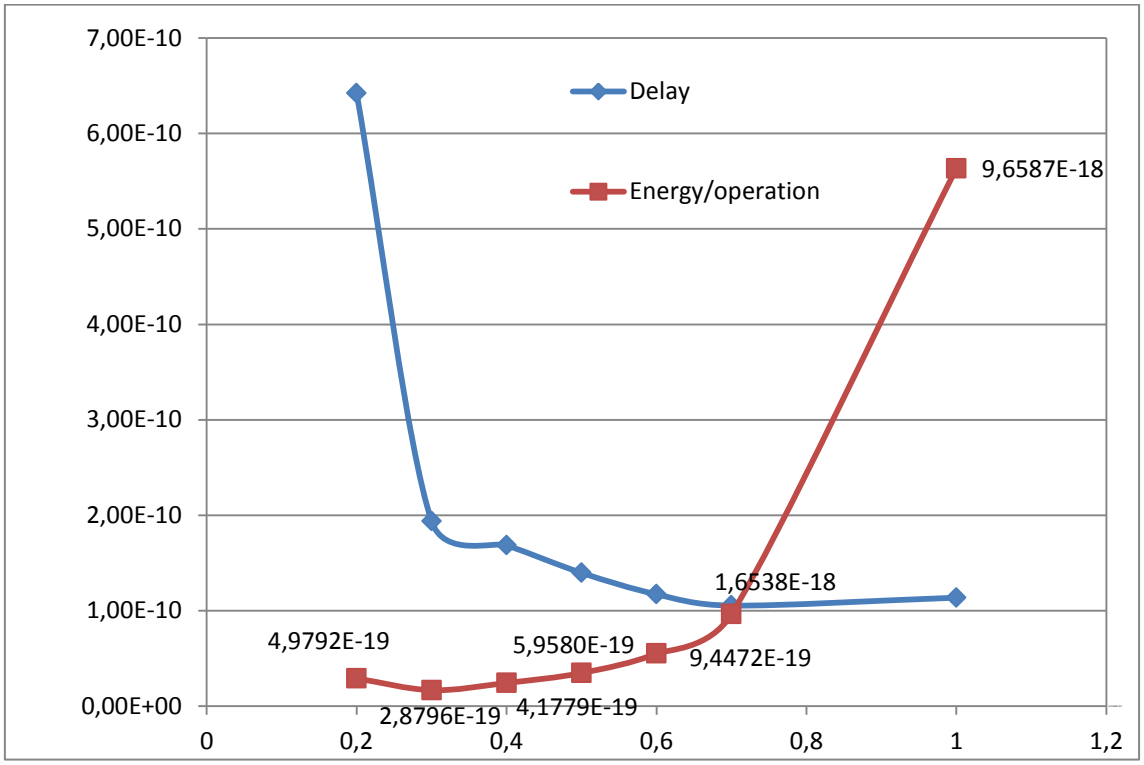


Figure 14, delay and energy/operation inverter without parasitics

The layout is created, see Figure 15.

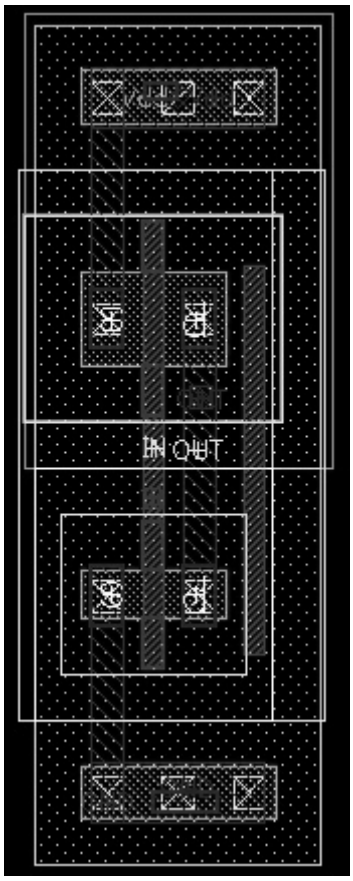


Figure 15, inverter layout

After the RTS and LVS test are passed, the parasitics are extracted and the data is calculated again, see table 6 and figure 16.

Table 7, inverter without parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average power	Δpower	Energy/operation	ΔEnergy/operation
0,2	1,08E-06	8,50E-07	1,62E-06	9200,0%	4,36E-08	0,7124%	7,0730E-14	65,5404%
0,3	1,96E-07	1,48E-07	2,94E-07	1667,2%	1,64E-07	2,6801%	4,8221E-14	44,6829%
0,4	6,57E-08	4,91E-08	9,86E-08	559,4%	2,94E-07	4,8016%	2,8987E-14	26,8602%
0,5	3,51E-08	2,51E-08	5,26E-08	298,6%	9,68E-07	15,8109%	5,0956E-14	47,2173%
0,6	2,37E-08	1,71E-08	3,55E-08	201,3%	1,45E-06	23,7139%	5,1510E-14	47,7305%
0,7	9,67E-09	1,14E-08	1,70E-08	96,6%	2,96E-06	48,3750%	5,0428E-14	46,7282%
1	1,18E-08	9,32E-09	1,76E-08		6,12E-06		1,0792E-13	

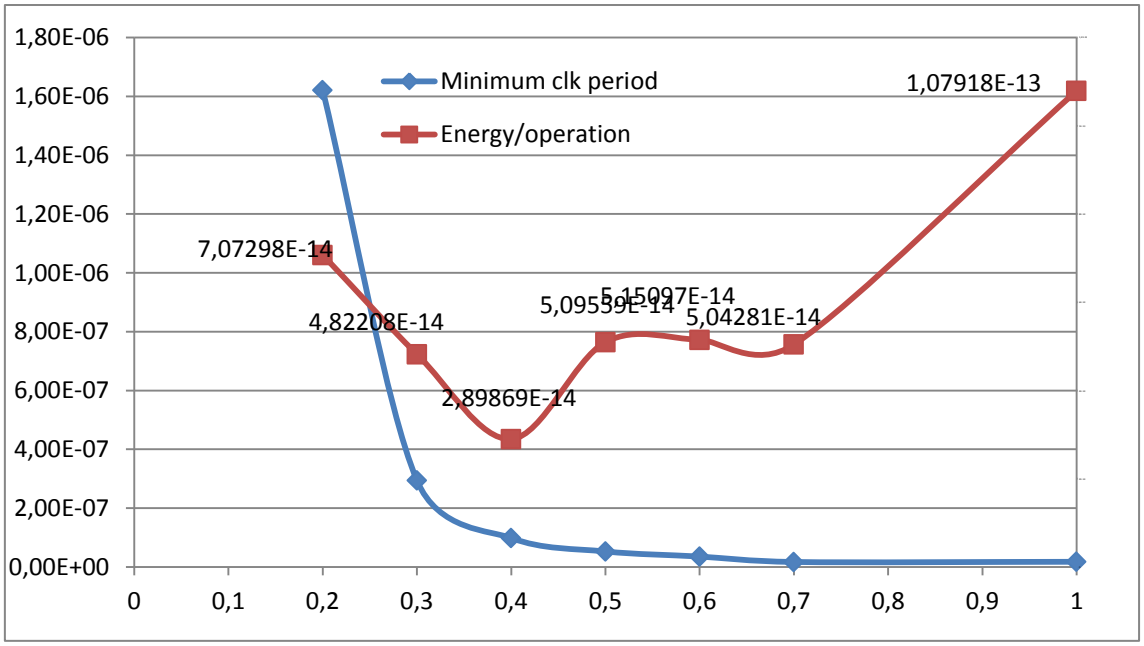


Figure 16, delay and energy/operation inverter with parasitics

3.2 NAND Gate

The data obtain in the NAND gate simulation is shown in table 8. The delay and energy per operation for the different voltages is shown in figure 13.

The worst delay in this gate happened when the signal had to change from 1 to 0. In this case the delay was conditioned by the fall time.

Table 8, NAND without parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average power	Δpower	Energy/operation	ΔEnergy/operation
0,2	3,99E-10	9,04E-10	1,36E-09	1088,7%	6,58E-10	1,6303%	8,9195E-19	17,7488%
0,3	1,46E-10	1,78E-10	2,67E-10	214,1%	1,26E-09	3,1219%	3,3585E-19	6,6831%
0,4	1,23E-10	1,21E-10	1,84E-10	147,9%	2,06E-09	5,1041%	3,7945E-19	7,5506%
0,5	1,04E-10	9,84E-11	1,56E-10	125,2%	3,14E-09	7,7800%	4,8937E-19	9,7379%
0,6	9,07E-11	8,28E-11	1,36E-10	109,3%	5,59E-09	13,8528%	7,6057E-19	15,1345%
0,7	8,03E-11	8,28E-11	1,24E-10	99,7%	8,28E-09	20,5154%	1,0283E-18	20,4610%
1	8,20E-11	8,30E-11	1,25E-10		4,04E-08		5,0254E-18	

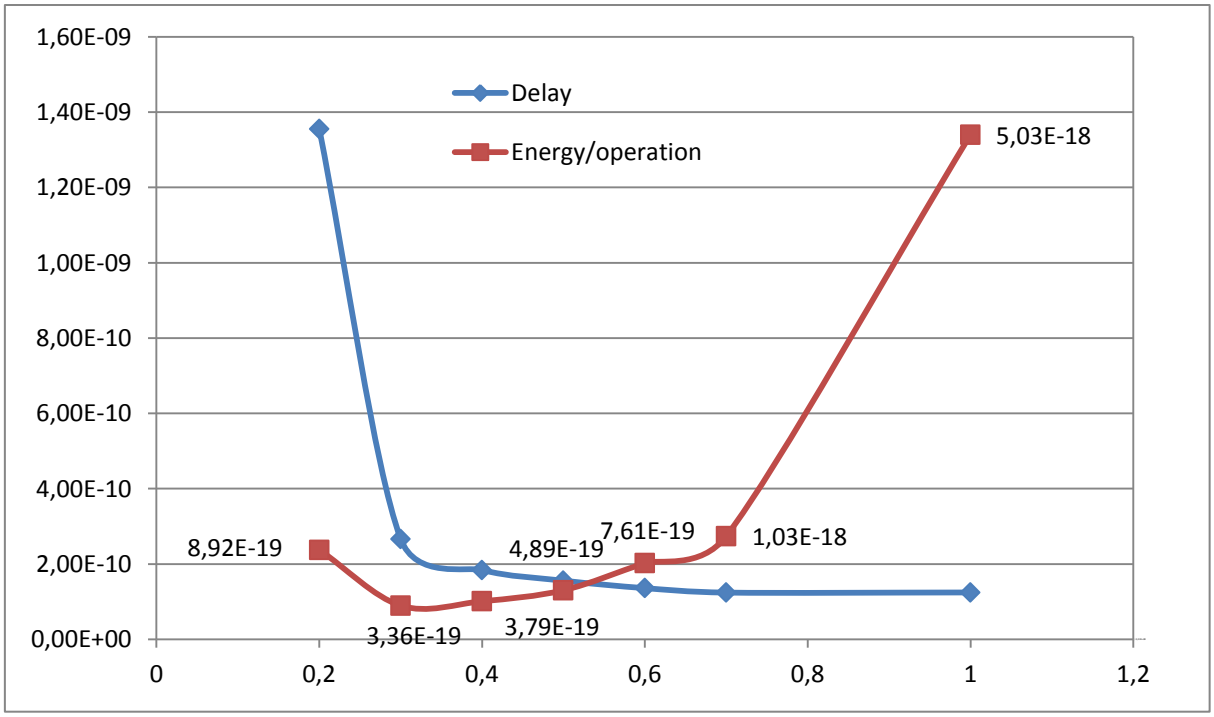


Figure 17, delay and energy/operation inverter without parasitics

The layout is created, see Figure 18.

After the RTS and LVS test are passed, the parasitics are extracted and the data is calculated again, see table 9 and figure 19.

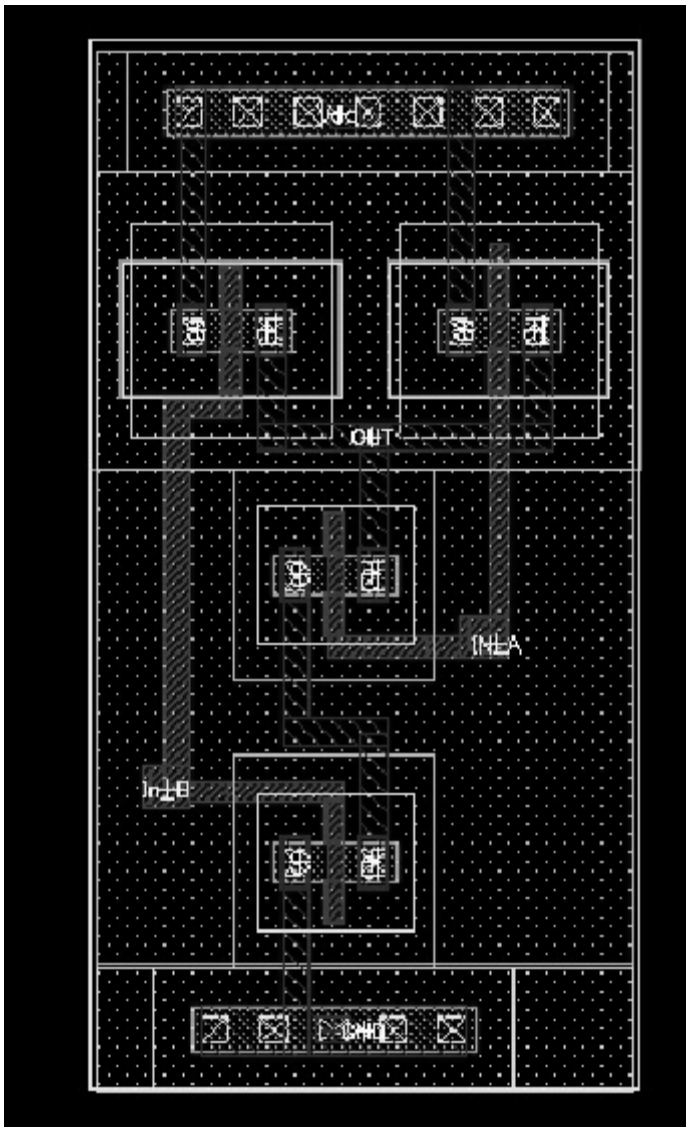


Figure 18, NAND layout

Table 9, NAND with parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average power	Δpower	Energy/operation	ΔEnergy/operation
0,2	6,52E-07	1,92E-06	2,87E-06	11785,2%	1,29E-09	0,0254%	3,7172E-15	2,9984%
0,3	1,28E-07	3,28E-07	4,92E-07	2017,2%	4,53E-08	0,8903%	2,2264E-14	17,9591%
0,4	4,86E-08	1,02E-07	1,53E-07	628,3%	8,21E-08	1,6144%	1,2575E-14	10,1436%
0,5	2,79E-08	5,05E-08	7,58E-08	311,0%	6,33E-07	12,4499%	4,7993E-14	38,7133%
0,6	1,95E-08	3,23E-08	4,84E-08	198,6%	2,26E-06	44,4554%	1,0944E-13	88,2815%
0,7	1,53E-08	2,48E-08	3,71E-08	152,4%	2,48E-06	48,8006%	9,2181E-14	74,3571%
1	1,09E-08	1,63E-08	2,44E-08		5,09E-06		1,2397E-13	

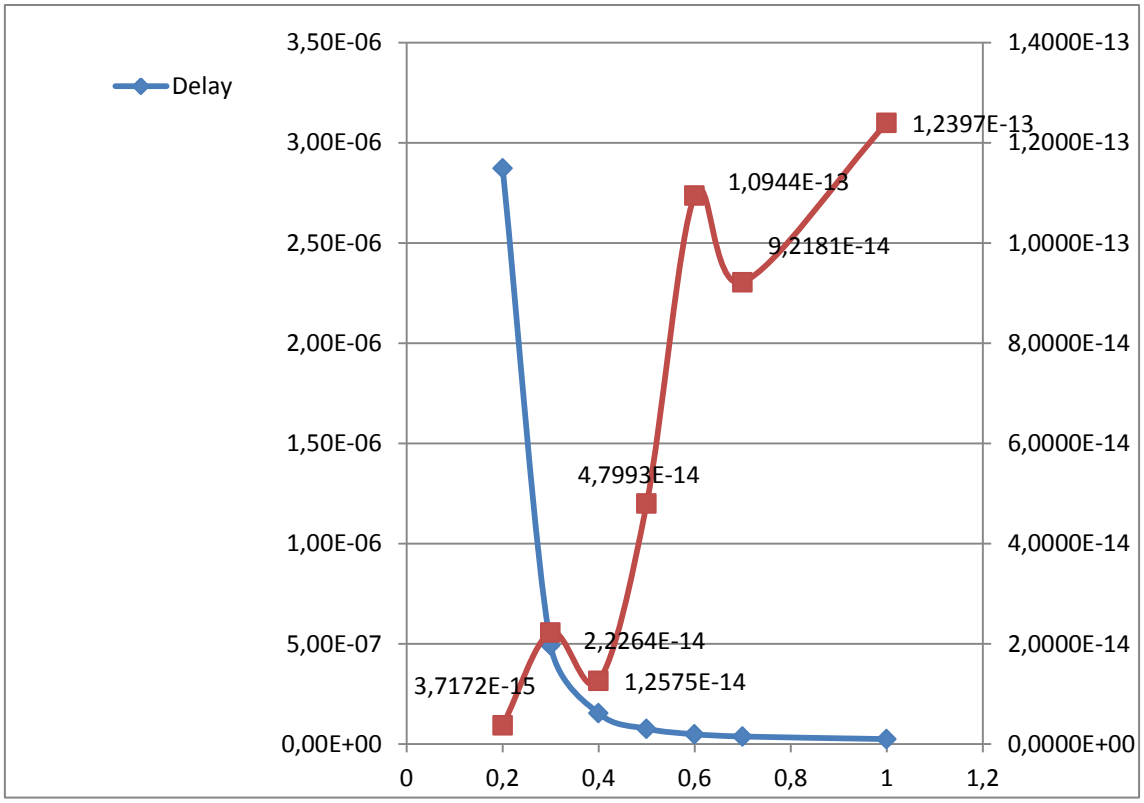


Figure 19, delay and energy/operation NAND with parasitics

3.3 NOR Gate

The data obtain in the NOR gate simulation is shown in table 10. The delay and energy per operation for the different voltages is shown in figure 20.

The worst delay in this gate happened when the signal had to change from 0 to 1. In this case the delay was conditioned by the fall time.

Table 10, NOR without parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average power	Δpower	Energy/operation	ΔEnergy/operation
0,2	1,49E-09	4,74E-10	2,23E-09	2018,8%	9,06E-10	0,8049%	2,0170E-18	16,2488%
0,3	3,14E-10	1,21E-10	4,71E-10	427,0%	2,06E-09	1,8302%	9,7010E-19	7,8150%
0,4	2,31E-10	1,03E-10	3,47E-10	314,2%	3,50E-09	3,1138%	1,2143E-18	9,7824%
0,5	1,87E-10	9,08E-11	2,81E-10	254,5%	5,97E-09	5,3031%	1,6753E-18	13,4957%
0,6	1,53E-10	7,81E-11	2,30E-10	208,1%	1,10E-08	9,7422%	2,5170E-18	20,2764%
0,7	1,37E-10	7,09E-11	2,06E-10	186,7%	2,09E-08	18,5689%	4,3023E-18	34,6589%
1	147,22-12	7,36E-11	1,10E-10		1,13E-07		1,2413E-17	

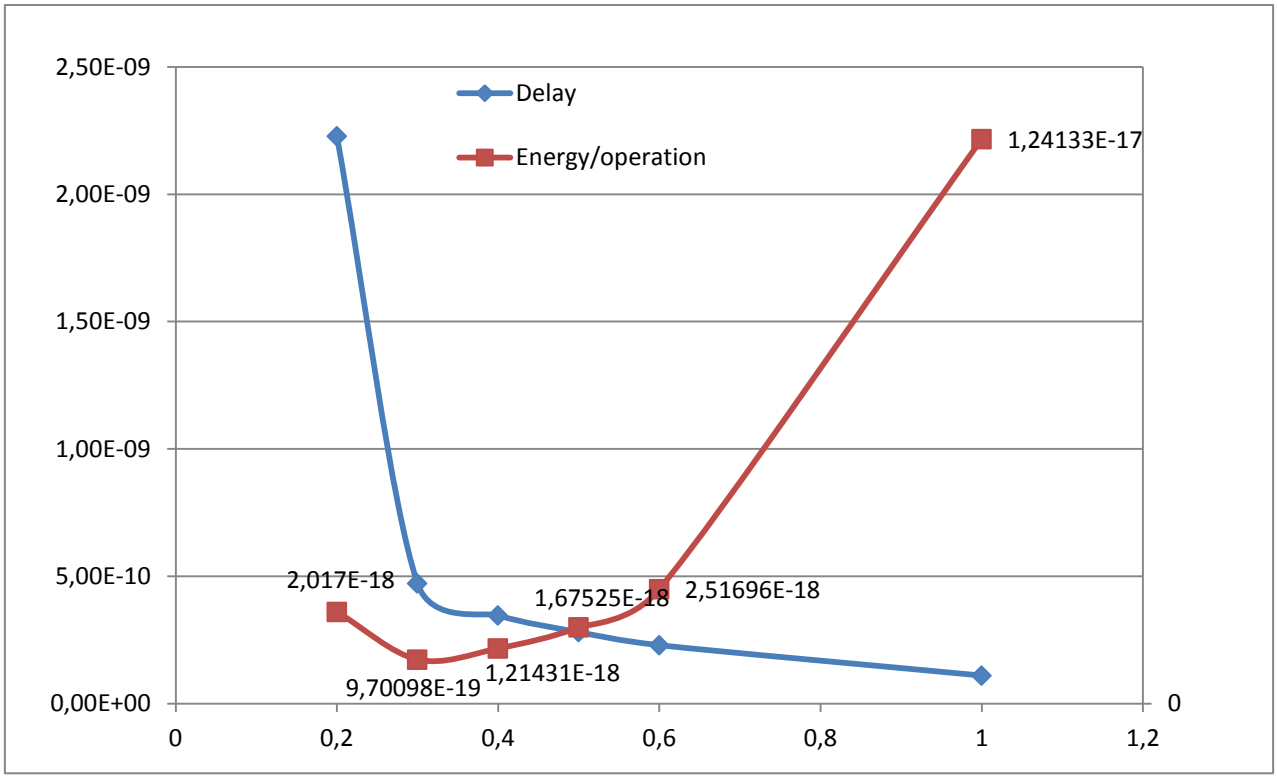


Figure 20, NOR delay and energy/operation inverter without parasitics

The layout is created, see Figure 22.

After the RTS and LVS test are passed, the parasitics are extracted and the data is calculated again, see table 10 and figure 23.

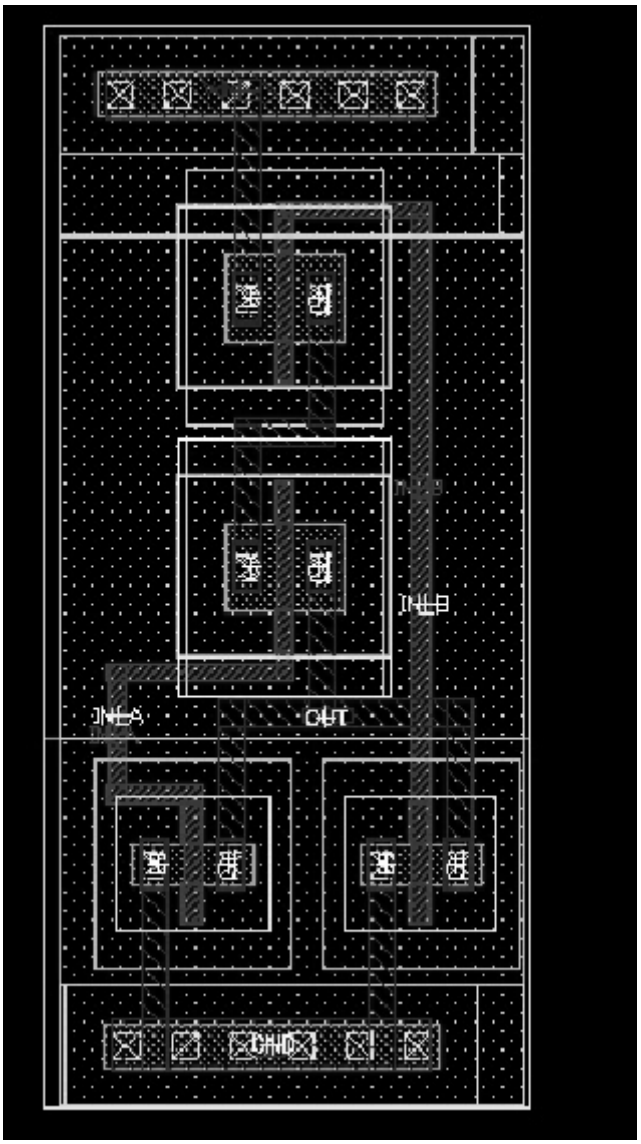


Figure 22, NOR layout

Table 10, NOR with parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average p	Δpower	Energy/operat	ΔEnergy/operation
0,2	2,42E-06	1,00E-06	3,64E-06	11767,2%	1,78E-09	0,0126%	6,4748E-15	1,4781%
0,3	2,75E-07	2,22E-07	4,13E-07	1337,3%	7,40E-08	0,5219%	3,0576E-14	6,9797%
0,4	9,14E-08	8,69E-08	1,37E-07	443,8%	1,40E-07	0,9849%	1,9148E-14	4,3710%
0,5	5,03E-08	4,66E-08	7,55E-08	244,4%	1,20E-06	8,4863%	9,0845E-14	20,7380%
0,6	3,28E-08	3,04E-08	4,93E-08	159,4%	4,43E-06	31,2640%	2,1830E-13	49,8324%
0,7	2,62E-08	2,12E-08	3,93E-08	127,2%	6,26E-06	44,1705%	2,4611E-13	56,1807%
1	2,06E-08	1,44E-08	3,09E-08		1,42E-05		4,3806E-13	

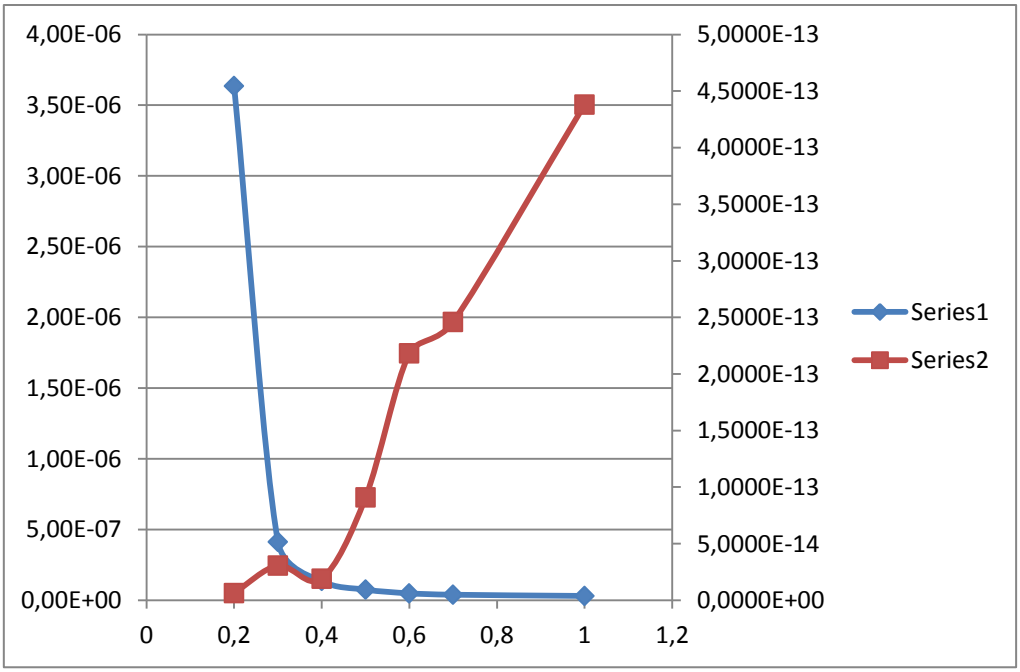


Figure 22 NOR delay and energy/operation inverter with parasitics

3.4 D-Flipflop

For the purpose of this project the propagation delay has been considered a ten percent of the clock period; the D signal has been set for a five percent of the clock cycle. This allows us to directly simulate the circuit with different frequencies to see when it start failing.

The data obtain in the flip-flop simulation is shown in table 11. The delay and energy per operation for the different voltages is shown in figure 23.

Table 11, D-Flipflop without parasitics data

Voltage	Minimum	Δminperiod	Average power	Δpower	Energy/operation	ΔEnergy/opera	Δmaxfrequency
0,2	4,90E-09	1088,7%	3,36E-09	0,0337%	8,3907E-17	0,3667%	9,2%
0,3	9,64E-10	214,1%	6,44E-09	0,3810%	1,8662E-16	0,8157%	46,7%
0,4	6,66E-10	147,9%	1,05E-08	2,2854%	7,7349E-16	3,3808%	67,6%
0,5	5,64E-10	125,2%	1,60E-08	7,4652%	2,1378E-15	9,3439%	79,9%
0,6	4,92E-10	109,3%	2,86E-08	17,5511%	4,3870E-15	19,1749%	91,5%
0,7	4,49E-10	99,7%	4,23E-08	30,0899%	6,8660E-15	30,0102%	100,3%
1	4,51E-10	100,0%	2,06E-07		2,2879E-14		100,0%

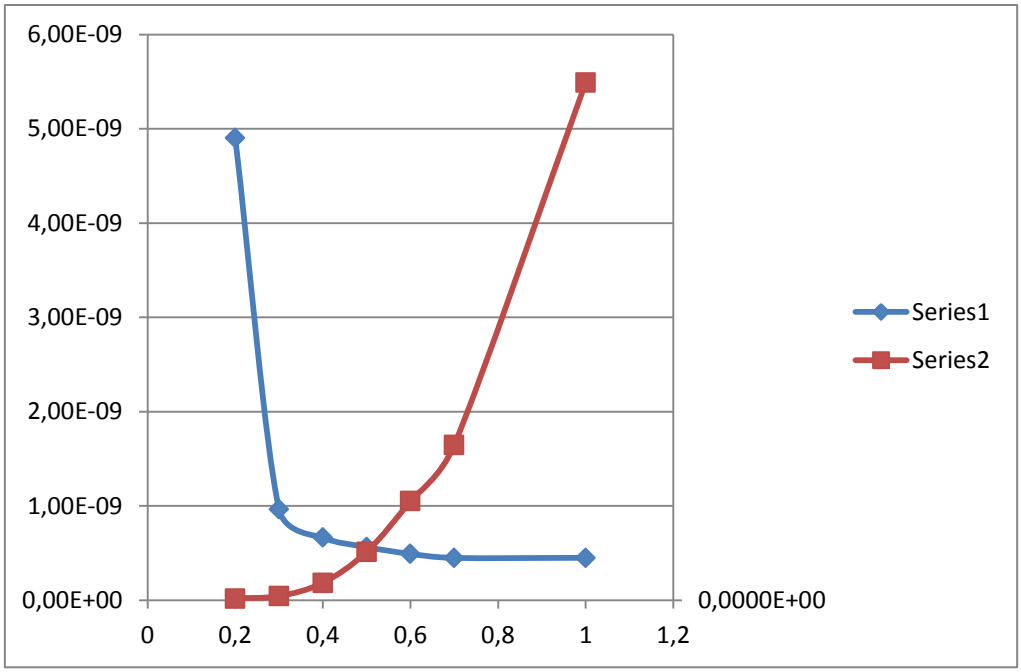


Figure 23, D-Flipflop delay and energy/operation inverter without parasitics

We simulate the D-Flipflop again using the version of the NAND and inverter with parasitics. The data obtain is shown in table 12. The delay and energy per operation for the different voltages is shown in figure 24.

Table 12, D-Flipflop with parasitics data

Voltage	Minimum	Δminperiod	Average power	Δpower	Energy/operation	ΔEnergy/operation	Δmaxfrequency
0,2	1,04E-05	11785,2%	6,61E-09	0,0%	6,8743E-14	3,0%	0,8%
0,3	1,78E-06	2017,2%	2,31E-07	0,9%	4,1173E-13	18,0%	5,0%
0,4	5,54E-07	628,3%	4,20E-07	1,6%	2,3255E-13	10,1%	15,9%
0,5	2,74E-07	311,0%	3,24E-06	12,4%	8,8755E-13	38,7%	32,2%
0,6	1,75E-07	198,6%	1,16E-05	44,5%	1,1538E-12	50,3%	50,4%
0,7	1,34E-07	152,4%	1,27E-05	48,8%	1,7047E-12	74,4%	65,6%
1	8,82E-08	100,0%	2,60E-05	100,0%	2,2926E-12	100,0%	100,0%

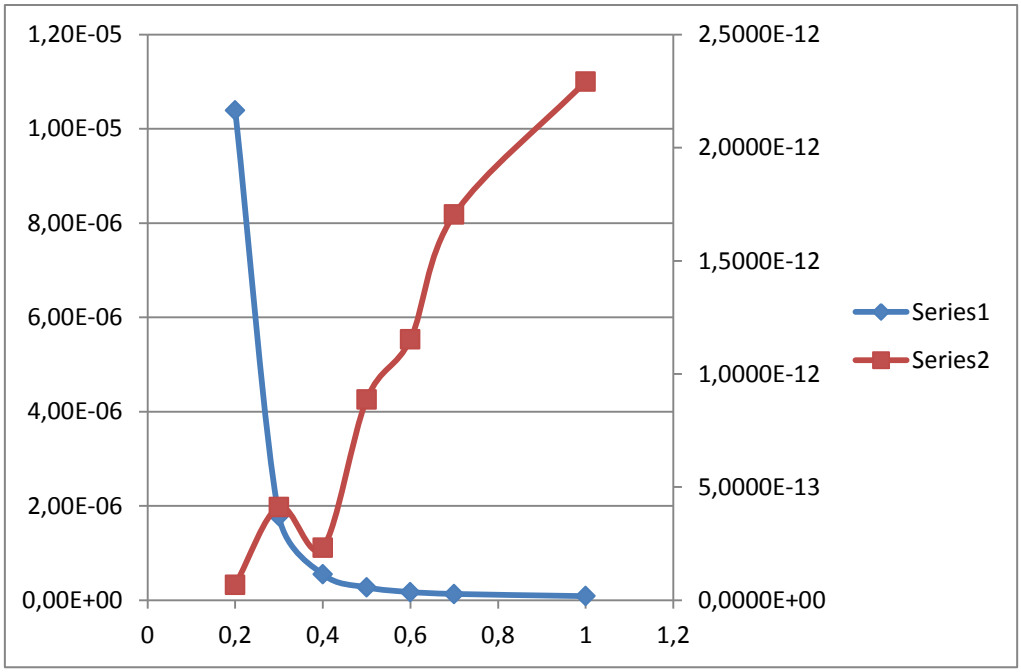


Figure 20, D-Flipfop delay and energy/operation inverter with parasitics

3.5 XOR

This gate did not exist in the previous library; it is an addition to improve the synthesis optimization. The results displayed are directly the ones obtain after extracting the parasitics from the layout, see table 13 and figure 25.

Table 13, XOR with parasitics data

Voltage	Rise time	Fall time	Delay	Δdelay	Average power	Δpower	Energy/operation	ΔEnergy/operation
0,2	3,82E-06	3,01E-06	5,73E-06	10906,1%	4,63E-08	0,1665%	2,6558E-13	18,1638%
0,3	5,07E-07	5,18E-07	7,77E-07	1478,6%	2,77E-07	0,9968%	2,1549E-13	14,7383%
0,4	1,69E-07	1,64E-07	2,54E-07	482,6%	5,08E-07	1,8247%	1,2875E-13	8,8056%
0,5	9,20E-08	8,22E-08	1,38E-07	262,6%	2,81E-06	10,0979%	3,8768E-13	26,5146%
0,6	6,08E-08	5,36E-08	9,11E-08	173,4%	8,24E-06	29,6027%	7,5071E-13	51,3436%
0,7	3,93E-08	3,93E-08	5,90E-08	112,3%	1,25E-05	45,0956%	7,4031E-13	50,6325%
1	3,50E-08	2,77E-08	5,25E-08		2,78E-05		1,4621E-12	

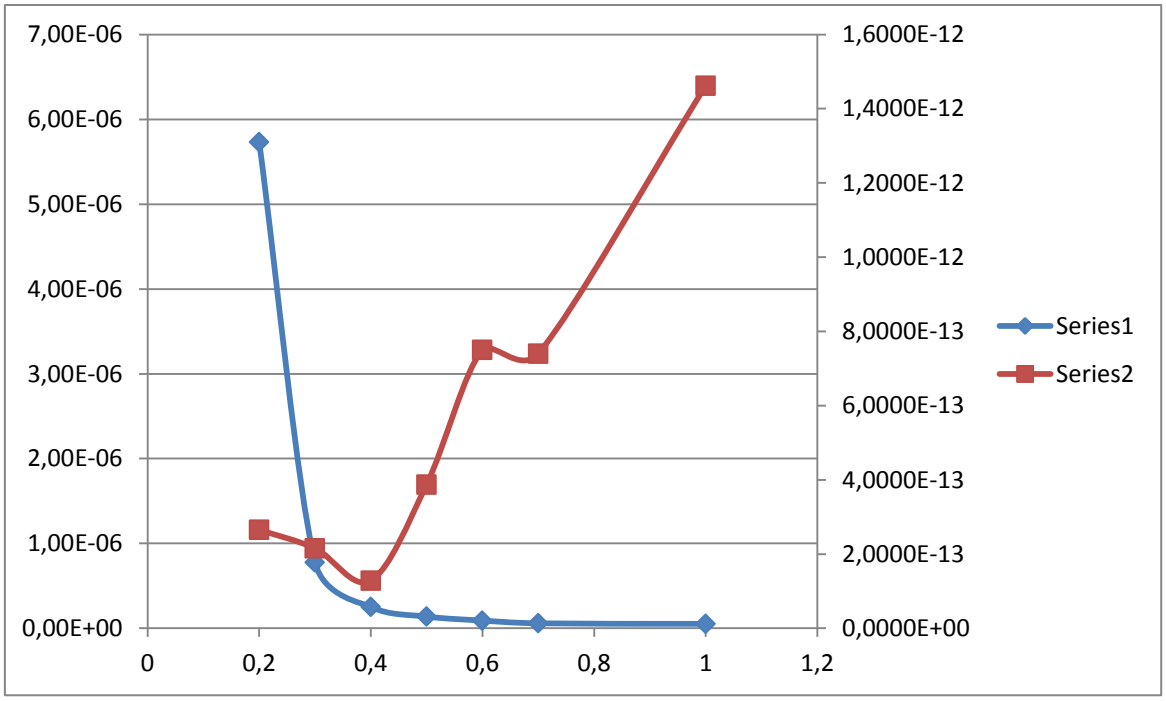


Figure 21, delay and energy/operation XOR with parasitics

3.6 Extraction

Now that all the cells in the library are created and their parasitics are extracted we proceed to extract the LEF file.

4 Synthesis process

The circuit synthesize in this thesis is Aurora's ALU. Once the RTL code has been extracted, we check the behaviour within our capabilities (doing a full verification is too hard in an ALU), see Figure 26. The check will consist in adding 2 numbers: 0011 0101 1001 0110 1111 1111 1111 1111$|_b$ and 0001 1110 1010 1010 0001 0001 0001 0001$|_b$ (these are 3596FFFF$|_h$ and 1EAA1111$|_h$). We assign them to the inputs a and b; then we select the addition operation by writing the out_sel to 0011$|_b$.

As expected, we obtain 0101 0100 0100 0001 0001 0001 0001 0000$|_b$ (54411110$|_h$) in the output (o_out).

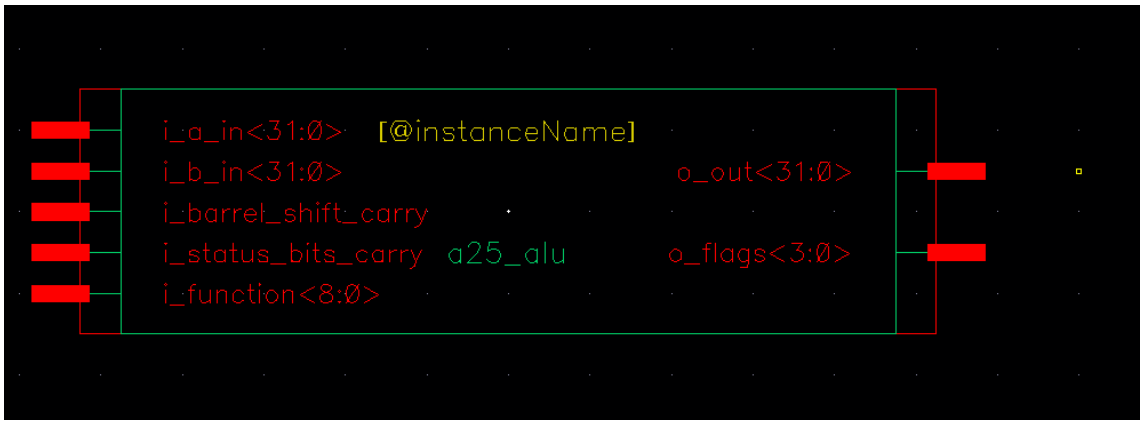


Figure 26, ALU symbol

With the LEF files and the RTL code we start the place and route. We export the DEF file. See Appendix 2.d.

The circuit pass both the DRC and the LVS.

Finally we simulate the ALU but including the parasitics. This time we select the worst delay case as operation, this will be and addition in which the carry has to ripple thought all the bits. The numbers: 1111 1111 1111 1111 1111 1111 1111 1111$|_b$ and 0000 0000 0000 0000 0000 0000 0000 0001$|_b$ (these are FFFFFFFF$|_h$ and 00000001$|_h$). We assign them to the inputs a and b; then we select the addition operation by writing the out_sel to 0011$|_b$. We obtain 0000 0000 0000 0000 0000 0000 0000 0000$|_b$ in o_out and a flag is raised for the overflow.

We proceed to measure the delay and the energy for this operation. The data is obtained for the ALU working at 1V, 0.5V, 0.4V and 0.3V. All operations have been made considering that the ALU is working at normal temperature (27ºC), see table 14.

Table 14, ALU results

Voltage	Delay	Δdelay	Average power	Energy/operation	ΔEnergy/operation
1V	8,01E-08		0,0026	2,08E-10	
0.5V	4,01E-07	5,01	0,000434724	1,74E-10	0,84
0.4V	1,06E-06	13,28	6,62789E-05	7,04E-11	0,34
0.3V	5,19E-06	64,93	6,94424E-05	3,61E-10	1,73

5 Discussion

5.1 Library

Including the parasitics in the gates has increase the delay of the cells and the power consumption in at least one order of magnitude. This proves that the parasitics effects should not be dismiss and it has proven especially significant for lower voltages.

5.1.1 Inverter

The inverter has the minimum energy/operation when working with four hundred milivolts supply (see figure 16 and table 7). A four hundred milivolts supply voltage would be the best operation point.

In this operation point:

- The energy per operations of the inverter is 26,86% of the energy per operation with one volt supply voltage.
- The delay is 559,4% of the delay with one volt supply voltage.

The cell may still be useful for near/subthreshold voltages supplies but the increase in the delay is significant and the reduction in energy per operation is not so significant.

5.1.2 NAND

The NAND gate has the minimum energy/operation when working with two hundred milivolts supply (see figure 19 and table 9). But in this point the delay has increased more than a hundred times and, despite the energy per operation been significantly lower, this delay is too big to work with. For this reason the best operation point would be a four hundred milivolts supply voltage.

In this operation point:

- The energy per operations of the NAND is 10,14% of the energy per operation with one volt supply voltage.
- The delay is 628,3% of the delay with one volt supply voltage.

The cell behavior in four hundred milivolts is pretty good being the reduction in energy per operation significantly bigger than the increase in the delay of the cell.

5.1.3 NOR

The NOR gate has the minimum energy/operation when working with two hundred milivolts supply (see figure 22 and table 10). But at this point the difference in energy per operation respect when working at three or four hundred milivolts is insignificant when compare with the huge increase in the delay. Four hundred milivolts is the next minimum in the energy per operation and has less delay than with three hundred milivolts, so we select four hundred milivolts as best operational point.

- The energy per operations of the NOR is 4,37 % of the energy per operation with one volt supply voltage.
- The delay is 443,8% of the delay with one volt supply voltage.

The cell behavior in four hundred milivolts is pretty good being the reduction in energy per operation significantly bigger than the increase in the delay of the cell.

5.1.4 D-Flipflop

The D-Flipflop has the minimum energy/operation when working with two hundred milivolts supply (see figure 24 and table 12). But in this point the maximum working frequency of the D-Flipflop is only 0,8% of the frequency when working using one volt power supply, despite the energy per operation been significantly lower. For this reason the best operation point would be a four hundred milivolts supply voltage.

In this operation point:

- The energy per operations of the D-Flipflop is 10,1% of the energy per operation with one volt supply voltage.
- The maximum frequency is 15,9% of the maximum frequency with one volt supply voltage.

The cell behavior in four hundred milivolts is acceptable been the decrease in energy per operation worth the decrease in maximum frequency.

5.1.5 XOR

The XOR gate has its minimum energy/operation when working with four hundred milivolts supply (see figure 25 and table 13). A four hundred milivolts supply voltage would be the best operation point.

In this operation point:

- The energy per operations of the XOR is 8,08% of the energy per operation with one volt supply voltage.

- The delay is 482,6% of the delay with one volt supply voltage.

The cell behavior in four hundred milivolts is pretty good being the reduction in energy per operation significantly bigger than the increase in the delay of the cell.

5.2 Synthesis

The results of simulating the synthesized ALU with parasitics for near threshold voltages are not as good as expected see table 14.

The best reduction in energy per operation occurs for four hundred millivolts, the voltage supply at which the library cells had a better behaviour. For this voltage supply:

- The energy per operation one third of the energy per operation with one volt voltage supply.
- The delay in the operation is 13,28 times the delay of the circuit with one volt power supply.

There may be many causes for these results:

- The use of only two inputs gates in the custom library make the synthesis tool use large areas (more gates) and increase the power consumption, including new cells in the library will improve both delay and power consumption but on the other hand it may cause the reliability of the circuit to decrease when working in the near/subthreshold area.
- The simulation was done for the worst case scenario, that means that the circuit will almost always have better results than showed.
- Cadence tools are very complex and include too many parameters. It was not possible to familiarize with all of them and there may be some that allows better optimization.
- 65nm technology has a lot of leakage and may not be the best fit for near/subthreshold operations.

6 Conclusions

During the course of this project a custom library cell has been design and optimized to work in the near/subthreshold area of operation.

This library has later on been used to synthesize an ALU. The results of simulating this ALU show that it is possible to reduce the power consumption of a circuit using near/subthreshold voltages when the speed of the circuit is not the main concern.

6.1 Further work

The possibilities for further work are large.

Further optimization of the library gates may still be possible. It would also be interesting to add some new cells to the library (multiple inputs, 1 bit adder…) and see how the new synthesized circuit behaves.

More work needs to be done to completely familiarize with the Cadence environment. This will allow to make use of its resources at its full potential.

7 References

[1] Analog integrated circuit design by Tony Chan Carusone, David Johns and Kenneth Martin. 2nd edition. Wiley

[2] Tutorial: Digital Logic Design and Verification Flow Using a Standard Cell Library by Amir Hasanbegovic. November 5. 2013.

[3] Coping with parametric varation at neat-threshold voltages by Ulya R. Karpuzcu

University of Minnesota, Nam Sung Kim University of Wisconsin-Madison and Josep Torrellas University of Illinois at Urbana-Champaign

[4] Near-Threshold Operation for Power-Efficient Computing? It Depends... by eland Chang

And Wilfried Haensch IBM T. J. Watson Research Center

[5] Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits by Ronald G. Dreslinski, Michael Wieckowski, David Blaauw, Senior Member IEEE,

Dennis Sylvester, Senior Member IEEE, and Trevor Mudge, Fellow IEEE.

[6] Prospects of Near-Threshold Voltage Design for Green Computing by Khare, S. and Jain, S.

Integrated Platforms Res., Intel Labs., Bangalore, India.

[7] Device Optimization for Ultra-Low Power Digital Sub-Threshold Operation by Bipul C Paul, Arijit Raychowdhury, and Kaushik Roy School of Electrical and Computer Engineering, Purdue University West Lafayette, IN 47907, USA.

[8] Design and analysis of metastable-hardened flip-flops in sub-threshold region by David Li, Pierce I-Jen Chuang, David Nairn and Manoj Sachdev. Department of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada

[9] TUTORIAL CADENCE DESIGN ENVIRONMENT by Antonio J. Lopez Martin Klipsch School of Electrical and Computer Engineering New Mexico State University October 2002.

[10] Multiobjective Optimization of an Ultra Low Voltage/Low Power Standard Cell Library for Digital Logic Synthesis by Martin Severin Orrebakken Haugland UNIVERSITY OF OSLO, Department of informatics.

[11] Open Digital HDL Synthesized Layout Flow for Mixed-Signal IC's by Martin James Robert Copus, The Ohio State University, 2003, USA

[12] Sharing Cadence Libraries

[13] https://vlsiwiki.soe.ucsc.edu/index.php/Main_Page

[14] http://community.cadence.com/cadence_technology_forums/f/31

[15] http://www.egr.msu.edu/classes/ece410/mason/files/TutorialB.pdf

[16] http://www.seas.upenn.edu/~ese570/manual_1.htm

[17] Synthesis and Timing Verification Tutorial By Dr. Ahmet Bindal

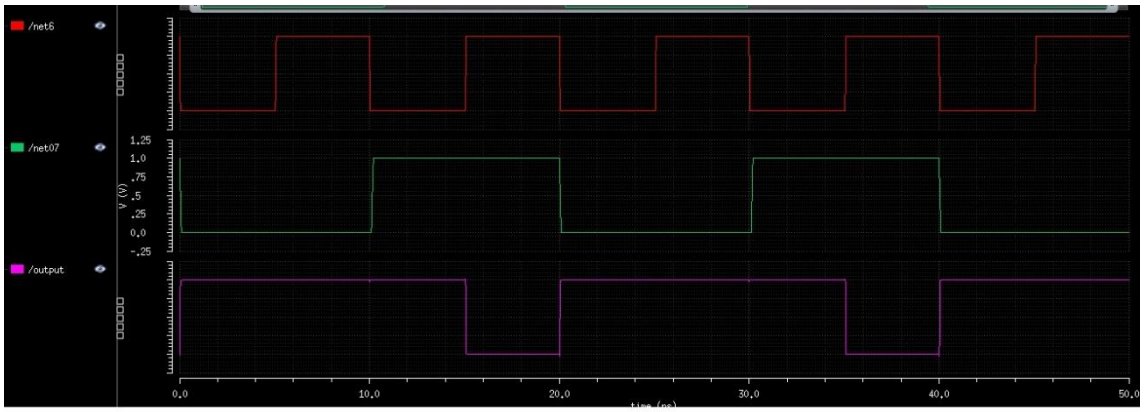
Computer Engineering Department of San Jose State University

[18] Encounter manual

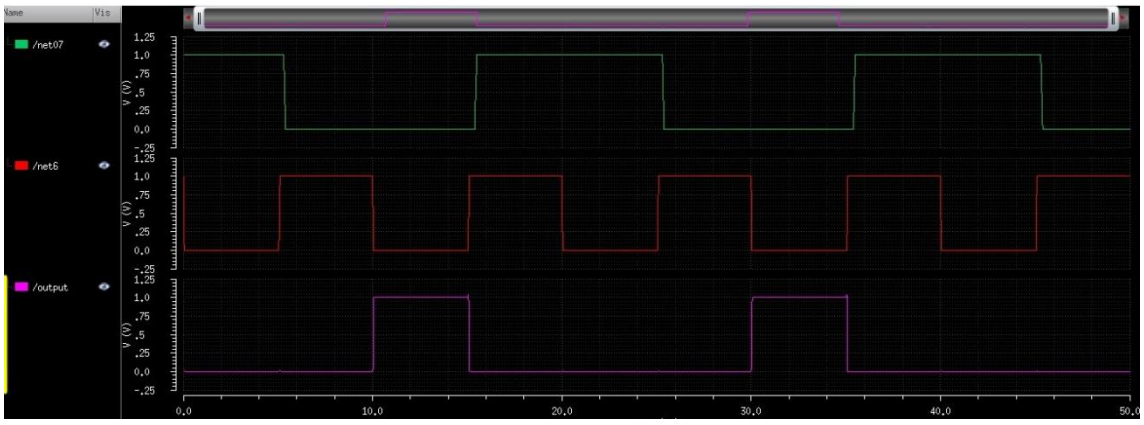
8 Appendix

1. GATE RESPONSE

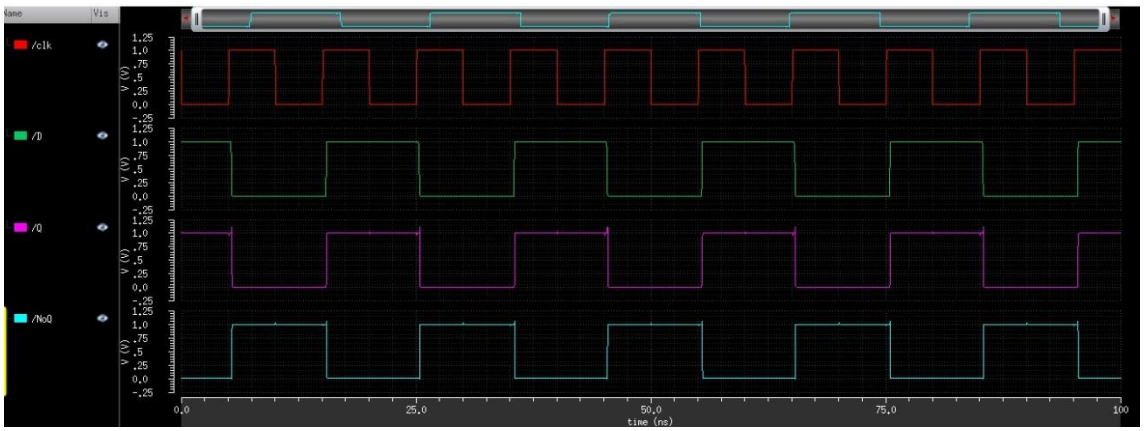
a. NAND gate response



b. NOR gate response



c. D Flip-flop response



2) Synthesis

a) Amber's ALU code (a25_alu.v)

```
//////////////////////////////////////////////////////////////////
//                                                              //
//  Arithmetic Logic Unit (ALU) for Amber 25 Core               //
//                                                              //
//  This file is part of the Amber project                      //
//  http://www.opencores.org/project,amber                      //
//                                                              //
//  Description                                                 //
//  Supported functions: 32-bit add and subtract, AND, OR,      //
//  XOR, NOT, Zero extent 8-bit numbers                         //
//                                                              //
//  Author(s):                                                  //
//      - Conor Santifort, csantifort.amber@gmail.com           //
//                                                              //
//////////////////////////////////////////////////////////////////
//                                                              //
// Copyright (C) 2011 Authors and OPENCORES.ORG                 //
//                                                              //
// This source file may be used and distributed without         //
// restriction provided that this copyright statement is not    //
// removed from the file and that any derivative work contains  //
// the original copyright notice and the associated disclaimer. //
//                                                              //
// This source file is free software; you can redistribute it   //
// and/or modify it under the terms of the GNU Lesser General   //
// Public License as published by the Free Software Foundation; //
// either version 2.1 of the License, or (at your option) any   //
// later version.                                               //
//                                                              //
// This source is distributed in the hope that it will be       //
// useful, but WITHOUT ANY WARRANTY; without even the implied   //
// warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR      //
// PURPOSE.  See the GNU Lesser General Public License for more //
// details.                                                     //
//                                                              //
// You should have received a copy of the GNU Lesser General    //
// Public License along with this source; if not, download it   //
// from http://www.opencores.org/lgpl.shtml                     //
//                                                              //
//////////////////////////////////////////////////////////////////


module a25_alu (

input      [31:0]         i_a_in,
```

```
input      [31:0]         i_b_in,
input                     i_barrel_shift_carry,
input                     i_status_bits_carry,
input      [8:0]          i_function,

output     [31:0]         o_out,
output     [3:0]          o_flags
);

wire    [31:0]        a, b, b_not;
wire    [31:0]        and_out, or_out, xor_out;
wire    [31:0]        sign_ex8_out, sign_ex_16_out;
wire    [31:0]        zero_ex8_out, zero_ex_16_out;
wire    [32:0]        fadder_out;
wire                  swap_sel;
wire                  not_sel;
wire    [1:0]         cin_sel;
wire                  cout_sel;
wire    [3:0]         out_sel;
wire                  carry_in;
wire                  carry_out;
wire                  overflow_out;
wire                  fadder_carry_out;

assign  { swap_sel, not_sel, cin_sel, cout_sel, out_sel } = i_function;


// ========================================================
// A Select
// ========================================================
assign a     = (swap_sel ) ? i_b_in : i_a_in ;

// ========================================================
// B Select
// ========================================================
assign b     = (swap_sel ) ? i_a_in : i_b_in ;

// ========================================================
// Not Select
// ========================================================
assign b_not     = (not_sel ) ? ~b : b ;

// ========================================================
// Cin Select
// ========================================================
assign carry_in  = (cin_sel==2'd0 ) ? 1'd0                :
                   (cin_sel==2'd1 ) ? 1'd1                :
                                      i_status_bits_carry    ;  // add with carry

// ========================================================
// Cout Select
// ========================================================
assign carry_out = (cout_sel==1'd0 ) ? fadder_carry_out     :
                                       i_barrel_shift_carry ;
```

```
// For non-addition/subtractions that incorporate a shift
// operation, C is set to the last bit
// shifted out of the value by the shifter.


// ======================================================
// Overflow out
// ======================================================
// Only assert the overflow flag when using the adder
assign  overflow_out    = out_sel == 4'd1 &&
                          // overflow if adding two positive numbers and get a negative
number
                        ( (!a[31] && !b_not[31] && fadder_out[31]) ||
                          // or adding two negative numbers and get a positive number
                          (a[31] && b_not[31] && !fadder_out[31])     );


// ======================================================
// ALU Operations
// ======================================================

`ifdef XILINX_FPGA

   // XIlinx Spartan 6 DSP module
   `ifdef XILINX_SPARTAN6_FPGA
      xs6_addsub_n #(.WIDTH(33))
   `endif
   `ifdef XILINX_VIRTEX6_FPGA
      xv6_addsub_n #(.WIDTH(33))
   `endif
      u_xx_addsub_33(
      .i_a    ( {1'd0,a}      ),
      .i_b    ( {1'd0,b_not}  ),
      .i_cin  ( carry_in      ),
      .i_sub  ( 1'd0          ),
      .o_sum  ( fadder_out    ),
      .o_co   (               )
   );

`else
assign fadder_out       = { 1'd0,a} + {1'd0,b_not} + {32'd0,carry_in};
`endif

assign fadder_carry_out = fadder_out[32];
assign and_out          = a & b_not;
assign or_out           = a | b_not;
assign xor_out          = a ^ b_not;
assign zero_ex8_out     = {24'd0,  b_not[7:0]};
assign zero_ex_16_out   = {16'd0,  b_not[15:0]};
assign sign_ex8_out     = {{24{b_not[7]}},  b_not[7:0]};
assign sign_ex_16_out   = {{16{b_not[15]}}, b_not[15:0]};

// ======================================================
// Out Select
// ======================================================
```

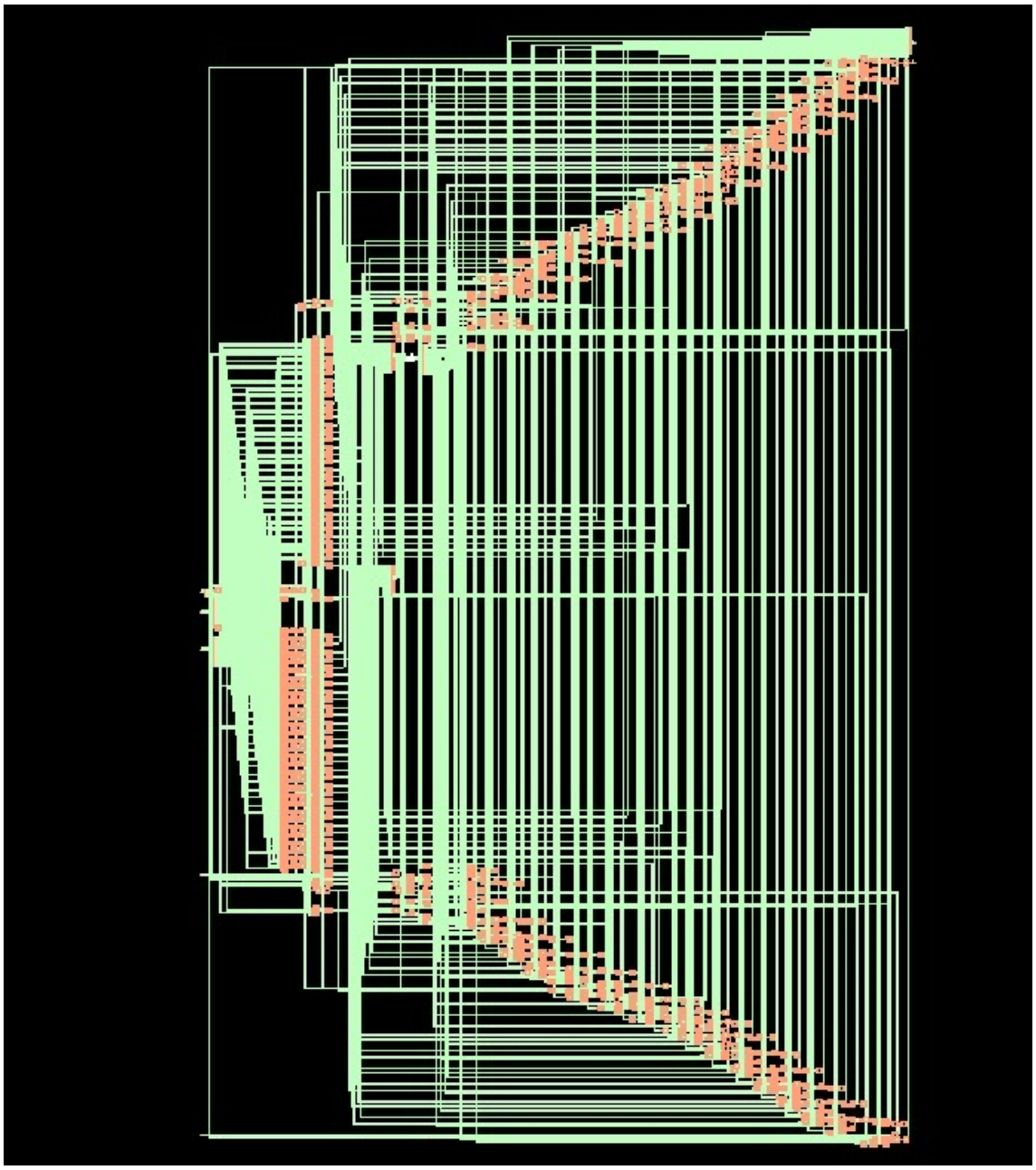
```
assign o_out = out_sel == 4'd0 ? b_not          :
               out_sel == 4'd1 ? fadder_out[31:0] :
               out_sel == 4'd2 ? zero_ex_16_out   :
               out_sel == 4'd3 ? zero_ex8_out     :
               out_sel == 4'd4 ? sign_ex_16_out   :
               out_sel == 4'd5 ? sign_ex8_out     :
               out_sel == 4'd6 ? xor_out          :
               out_sel == 4'd7 ? or_out           :
                                 and_out          ;

assign o_flags       = {  o_out[31],      // negative
                        |o_out == 1'd0,  // zero
                        carry_out,       // carry
                        overflow_out     // overflow
                        };


endmodule
```

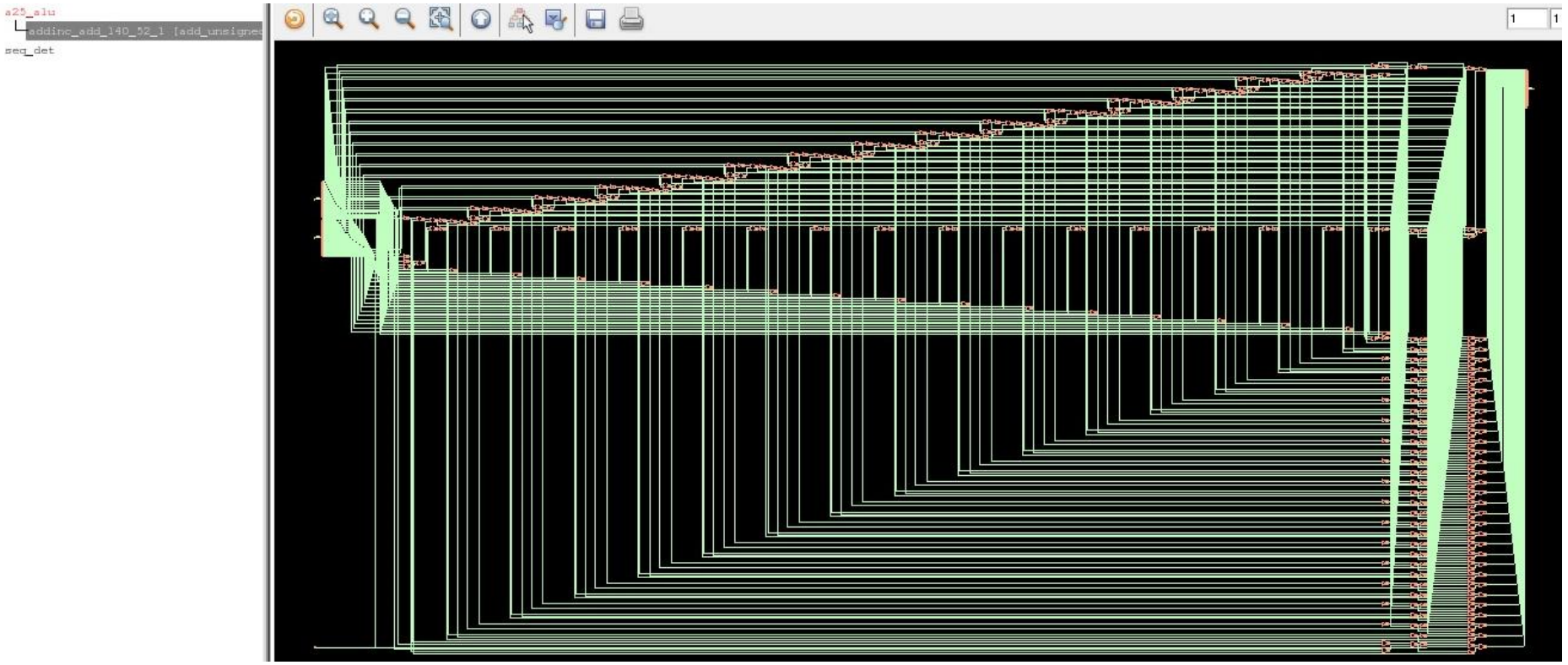
b) RTL code

```
// Generated by Cadence Encounter(R) RTL Compiler v11.20-s017_1

// Verification Directory fv/a25_alu

module add_unsigned_carry(A, B, CI, Z);
  input [31:0] A, B;
  input CI;
  output [32:0] Z;
  wire [31:0] A, B;
  wire CI;
  wire [32:0] Z;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
  wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
  wire n_32, n_33, n_34, n_35, n_36, n_37, n_38, n_39;
  wire n_40, n_41, n_42, n_43, n_44, n_45, n_46, n_47;
  wire n_48, n_49, n_50, n_51, n_52, n_53, n_54, n_55;
  wire n_56, n_57, n_58, n_59, n_60, n_61, n_62, n_63;
  wire n_64, n_65, n_66, n_67, n_68, n_69, n_70, n_71;
  wire n_72, n_73, n_74, n_75, n_76, n_77, n_78, n_79;
  wire n_80, n_81, n_82, n_83, n_84, n_85, n_86, n_87;
  wire n_88, n_89, n_90, n_91, n_92, n_93, n_94, n_95;
  wire n_96, n_97, n_98, n_99, n_100, n_101, n_102, n_103;
  wire n_104, n_105, n_106, n_107, n_108, n_109, n_110, n_111;
  wire n_112, n_113, n_114, n_115, n_116, n_117, n_118, n_119;
  wire n_120, n_121, n_122, n_123, n_124, n_125, n_126, n_127;
  wire n_128, n_129, n_130, n_131, n_132, n_133, n_134, n_135;
  wire n_136, n_137, n_138, n_139, n_140, n_141, n_142, n_143;
  wire n_144, n_145, n_146, n_147, n_148, n_149, n_150, n_151;
  wire n_152, n_153, n_154, n_155, n_156, n_157, n_158, n_159;
  wire n_160, n_161, n_162, n_163, n_164, n_165, n_166, n_167;
  wire n_168, n_169, n_170, n_171, n_172, n_173, n_174, n_175;
  wire n_176, n_177, n_178, n_179, n_180, n_181, n_182, n_183;
  wire n_184, n_185, n_187, n_188, n_189, n_191, n_192, n_193;
  wire n_194, n_195, n_197, n_198, n_199, n_200, n_201, n_203;
  wire n_204, n_205, n_206, n_207, n_209, n_210, n_211, n_212;
  wire n_213, n_215, n_216, n_217, n_218, n_219, n_221, n_222;
  wire n_223, n_225, n_226, n_227, n_228, n_229, n_231, n_232;
```

```
wire n_233, n_234, n_235, n_237, n_238, n_239, n_240, n_241;
wire n_242, n_243, n_245, n_246, n_247, n_248, n_249, n_251;
wire n_252, n_253, n_254, n_255, n_257, n_258, n_259, n_260;
wire n_261, n_263, n_264, n_265, n_266, n_267, n_269, n_270;
wire n_271, n_272, n_273, n_275, n_276, n_277, n_279, n_280;
wire n_281, n_282, n_283, n_285, n_286, n_287, n_288, n_289;
wire n_291, n_292, n_293, n_294, n_295, n_297, n_298, n_299;
wire n_300, n_301, n_303, n_304, n_305, n_306, n_307, n_308;
wire n_309, n_311, n_312, n_313, n_315, n_316, n_317, n_318;
wire n_319, n_321, n_322, n_323, n_324, n_325, n_327, n_328;
wire n_329, n_330, n_331, n_333, n_334, n_335, n_336, n_337;
wire n_338, n_339, n_341, n_342, n_343, n_345, n_346, n_347;
wire n_348, n_349, n_351, n_352, n_353, n_354, n_355, n_357;
wire n_358, n_359, n_360, n_361, n_363, n_364, n_365, n_366;
wire n_367;
NAND2X1 g2244(.A (n_367), .B (n_81), .Z (Z[32]));
NAND2X1 g2245(.A (n_365), .B (n_366), .Z (Z[31]));
NAND2X1 g2246(.A (n_364), .B (n_8), .Z (n_367));
NAND2X1 g2247(.A (n_364), .B (n_149), .Z (n_366));
NAND2X1 g2248(.A (n_363), .B (n_150), .Z (n_365));
INVX1 g2249(.A (n_363), .Z (n_364));
NOR2X1 g2250(.A (n_70), .B (n_361), .Z (n_363));
```

c) Synthesized schematic ALU and adder.




a25_alu
addinc_add_140_52_1 [add_unsigned
seq_det

d. Final layout

3) Layout

a) Layout optimization flow