# Power estimation using Design Compiler and Nanosim

## Description

This document is intended to serve as a manual for laboratory 1 exercises in the course TSTE85, low power electronics, provided at the division of Integrated Circuits and Systems, dept. of Electrical Engineering, Linköping University.

## Contents

# 1 Introduction

The purpose of this laboratory is mainly to introduce tools used for implementation of low power designs using VHDL, logic synthesis and standard cells. The tools shall also be used in the project part of this course. The following tools are discussed in this manual.

- Modelsim from Mentor Graphics for VHDL simulation
- Design Compiler from Synopsys for logic synthesis and power estimations
- Nanosim from Synopsys for power estimations

This document is divided into four small parts: a very short introduction to VHDL, a Modelsim tutorial, a Design Compiler tutorial and a tutorial on Nanosim.

## 1.1 Example design

The example used in the first part of this tutorial is a 4-bit ripple carry adder with registers at the inputs and at the outputs as shown in Fig. 1.

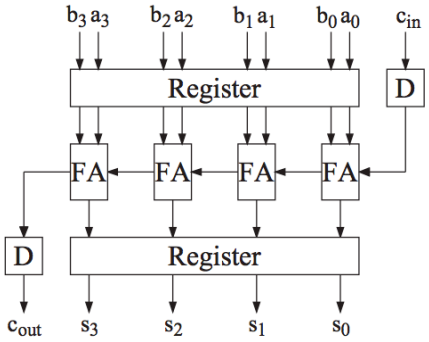


Figure 1: Four bit ripple carry adder.

# 2 A short introduction to VHDL

For the exercises we shall use structural VHDL coding, which is a small subset of the language. In this chapter a brief overview of how to write structural VHDL code is provided. The student is encouraged to read further about the language for more details.

## 2.1 Some basic VHDL syntax

- VHDL is not case sensitive.
- Semicolon is used to terminate a statement.
- Two dashes ('- -') are used to comment.
- Identifiers must begin with an alphabet; subsequent characters can include alphanumeric characters or '_' (underscore).

## 2.2 Some useful constructs in VHDL

A signal assignment is done using the <= operator. The signal to be assigned a value is placed on the left hand side and the value is placed on the right hand side. The value can be either a constant, another signal, or an expression as shown below.

```
a <= '1';
b <= c;
c <= d AND e;
```

The last example includes an AND statement. This corresponds to the logical AND operation between, in this case, the signals d and e. The result is assigned to a signal c. Commonly used logical operations such as AND, XOR, OR, NAND, NOR, XNOR and NOT are available in the language. VHDL is a concurrent programming language. If we consider the three signal assignments presented above, the concurrency implies that all the three assignments are performed in parallel. For example, if d changes from '0' to '1' while e is '1', c shall transit from '0' to '1'. At the same time b shall change from '0' to '1'. Operations can be executed in sequence as well. For this a process statement is introduced. Any code written inside a process statement is executed in sequence. A typical sequential construct is the if-statement. The signals that should activate the process is collected in the sensitivity list.

```
process(<sensitivity_list>)
if ( condition ) then ....
end if | elsif ....
end process;
```

## 2.3 Library declaration

The LIBRARY statement is used to reference a group of previously defined VHDL design units, other entities or groups of procedures/functions together known as packages. The USE statement specifies what entities or packages to use out of this library; in this case USE IEEE.std_logic_1164.all imports all procedures/functions in the std_logic_1164 package. The 1164 single bit type std_logic and the vector type std_logic_vector (for busses) shall be used for all signal types in the tutorial examples.

## 2.4 Examples of combinational circuits

Some basic components useful in this course are described in structural VHDL in this section.

### 2.4.1 Full adder

Full adder is a commonly used arithmetic circuit.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fulladder is
port(
  signal a, b, cin: in std_logic;
  signal sum, cout: out std_logic);
end fulladder;

architecture structural of fulladder is
begin
  sum <= a XOR b XOR cin;
  cout <= (a AND b) OR (a AND cin) OR (b AND cin);
end structural;
```

### 2.4.2 1-bit multiplexer

A 2-1 multiplexer can be described using a process and an if-statement. On the sensitivity list we have the three input signals. Each time anyone of these signals has a transition, the process is executed.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity two2one_mux is
port(
  signal a, b, sel: in std_logic;
  signal o: out std_logic);
end two2one_mux;

architecture structural of two2one_mux is
begin
  process(a, b, sel)
  begin
  if sel = '0' then
    o <= a;
  else
    o <= b;
  end if;
  end process;
end structural;
```

In the above examples signals with a width of one bit have been used. The signals are of the type std_logic. However, it is also possible to use bit vectors which will be shown in further examples.

### 2.4.3 4-bit multiplexer

The design is a 2 to 1 multiplexer with a data width of 4 bits. Here we use std_logic_vector(width 1 downto 0) as the data type.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity two2one_mux_4bit is
port(
  signal a, b: in std_logic_vector(3 downto 0);
  signal sel: in std_logic;
  signal o: out std_logic_vector(3 downto 0));
end two2one_mux_4bit;

architecture structural of two2one_mux_4bit is
begin
  process(a, b, sel)
  begin
  if sel = '0' then
    o <= a;
  else
    o <= b;
  end if;
  end process;
end structural;
```

### 2.4.4 New component constructed using components

In this example, it is shown how a design can be constructed using several instances of a component. The full adder introduced above is used to construct a 4 bit ripple carry adder (in vhdl-files/ripple_carry_adder.vhdl). You will use components later on in the project.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity ripple_carry_4bit is
port(
  signal a, b: in std_logic_vector(3 downto 0);
  signal cin: in std_logic;
  signal cout: out std_logic;
  signal sum: out std_logic_vector(3 downto 0));
end ripple_carry_4bit;

architecture structural of ripple_carry_4bit is

component fulladder
port(
  signal a, b, cin: in std_logic;
  signal sum, cout: out std_logic);
end component;

signal c_1, c_2, c_3 : std_logic;

begin
  FA_0: fulladder portmap (a(0), b(0), cin, sum(0), c_1);
  FA_1: fulladder portmap (a(1), b(1), c_1, sum(1), c_2);
```

```
  FA_2: fulladder portmap (a(2), b(2), c_2, sum(2), c_3);
  FA_3: fulladder portmap (a(3), b(3), c_3, sum(3), cout);
end structural;
```

### 2.4.5 Example of a sequential circuit

A basic sequential component is the D-flip-flop. Here, the code for a positive edge triggered D flip-flop with an asynchronous reset is shown.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity DFF is
port(
  signal d, clk, reset: in std_logic;
  signal q: out std_logic);
end DFF;

architecture structural of DFF is
begin
  process(clk, reset, d)
  begin
  if reset = '1' then
    q <= '0';
  elsif rising_edge(clk) then
    q <= d;
  end if;
  end process;
end;
```

## 2.5 Further reading

The above section is merely a quick-start overview to VHDL in order to help the student through the exercises and the project. For further details, the student is encouraged to utilize the abundant resources on the web as well as several books available on the topic. A collection of links to VHDL websites suitable for beginners can be found at http://www.1001tutorials.com/vhdl/index.shtml. A book for further study of VHDL for synthesis: "VHDL for Programmable Logic" by Kevin Skahill.

# 3 Setting up the laboratory

Open a terminal window and start by logging onto the server naum via the intermediate server ssh with the commands

ssh -X ssh.edu.liu.se
ssh -X naum.ad.liu.se

Go to your home directory by typing

```
cd $HOME
```

on the command line. Now you need to create a course directory and a directory for this lab.

```
mkdir TSTE85
cd TSTE85
mkdir lab1
cd lab1
```

Copy the directories for this lab

```
cp -r /coop/e/eks/course/TSTE85/lab1/files/* .
```

Take a look on what has been copied

```
ls
```

You should find five directories (Design Analyzer, Design Compiler, Modelsim, Nanosim, vhdl-files) in the current directory. Load the modules for this lab:

```
module load mentor
module load synopsys/2011.09
```

Keep the terminal window open, since you will use it throughout this lab.

# 4 Introduction to Modelsim

We will use the 4-bit ripple carry adder that is described in section 1.1 as an example.

1. Ensure that the current location is the Modelsim directory using the `pwd` command. Use the command `cd` to call the right directory.
2. Create a directory for the vhdl compiler. Note that the command vlib must be used, not `mkdir`.

   ```
   vlib work
   ```

3. Compile the file `ripple_carry_adder.vhdl` using the commands:

   ```
   vcom -93 ../vhdl-files/ripple_carry_adder.vhdl
   ```

4. Start the Modelsim simulator using the command:

   ```
   vsim &
   ```

5. In the ModelSim window, on the left hand side, a library is shown. Click on the cross to the left of the work folder. Double click on `ripple_carry_4bit`.
6. The Wave window displays waveforms, names and current values for signals you have selected.

   ```
   view wave
   ```

7. To add the signals to the Wave window, right-click on the 'Objects' window and select Add to Wave... / Signals in region, from the dialog box.
8. Apply stimulus to the clock input in the Transcript prompt:

   ```
   force clk 0 0, 1 2 -repeat 4
   ```

   The force command is case-insensitive. The above command is interpreted by Modelsim to mean:

   - Force clk to '0' at 0 ns after the current time
   - Force clk to '1' at 2 ns after the current time
   - Repeat the cycle every 4 ns

   The other inputs can be assigned in a similar way.

   ```
   force a 1011
   force b 1001
   force cin 0
   force reset 0
   ```

   You will see the effects of the force commands as soon as you tell the simulator to run.
9. Run for 16 ns.

   ```
   run 16
   ```

10. Right-click in the wave window and select Zoom Full.
11. After how many clock cycles are the respective signals valid?

    ---

    ---

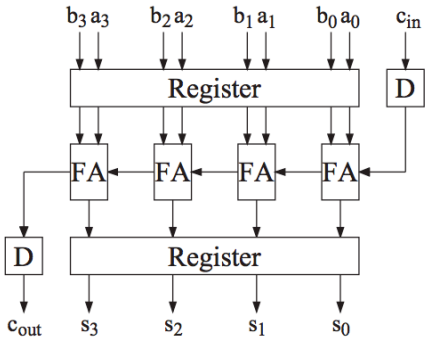


Figure 2: Four bit ripple carry adder.

12. What is the final output result?

    ---

13. Determine and sketch where the internal signals are localized in the figure below.

14. Now it is time to test the functionality of the adder using the simulator. Test a few different input values of a and b and verify the functionality. Which inputs did you use and which results did you get?

    ---

    ---

    ---

    ---

## 4.1 Useful tips in Modelsim

Here are some hints that might be useful during the labs and the project.

`force a 10#5` - same as `force a 0101`. Simplifies if the signal a is very wide.

Copy and paste text commands from the transcript file. All items that do not start with `#` is a vsim simulator command.

To change the radix of a signal in the wave window:
1. Select the signal name in the wave window.
2. Click right mouse button, choose radix and select appropriate radix, eg. decimal

# 5 Introduction to Design Compiler

Design Compiler from Synopsys is a tool for logic synthesis. It takes design written in VHDL or Verilog as input and translates it to a netlist consisting of logical building blocks. The building blocks can be, for example, the logical cells of an FPGA or standard cells in a library. In this laboratory we us a standard cell library implemented in a 0.35 $\mu$m CMOS process.

## 5.1 Logic synthesis using Design Compiler

1. Make sure that the current directory in the terminal window is DesignCompiler. Use command `cd` to navigate, if required. Start the graphical interface for Design Compiler using the command:

   ```
   design_vision &
   ```

   A window similar to that shown in Fig. 3 should appear.

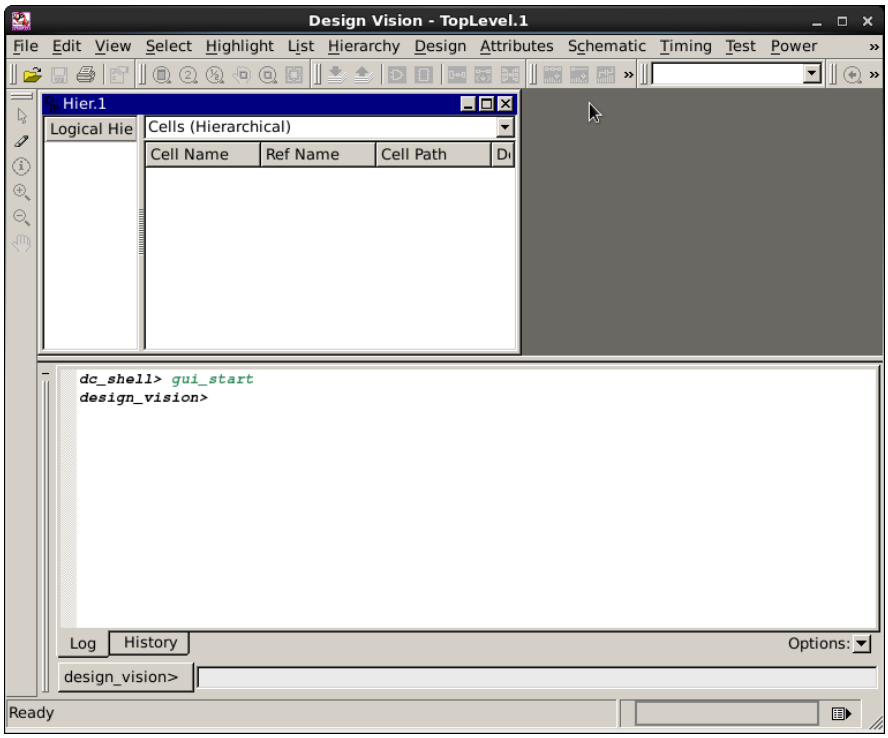


Figure 3: Design Compiler window

2. Now we shall use the command line interface of the Design Compiler, seen at the bottom of the window. Create a proper WORK-directory by entering the following commands in the command line interface of Design Compiler.

   ```
   sh mkdir WORK
   define_design_lib WORK -path "./WORK"
   ```

3. The next step is to read the vhdl file(s) into the tool. This is done by the command `analyze`.

   ```
   analyze -format vhdl -lib WORK ../vhdl-files/ripple_carry_adder.vhdl
   ```

If several files are to be analyzed, these can be specified simultaneously by <path1> <path2>.

4. The next step, `elaborate`, maps the vhdl code to generic logic building blocks. The top cell in the design described by `ripple_carry_adder.vhdl` is `ripple_carry_4bit` which should be specified together with the elaborate command.

   ```
   elaborate ripple_carry_4bit -arch "structural" -lib WORK -update
   ```

   Now, the building blocks of the design appears in the main window as shown in Fig. 4. The graphical display of the design can be enabled by clicking on the button marked in Fig. 4.

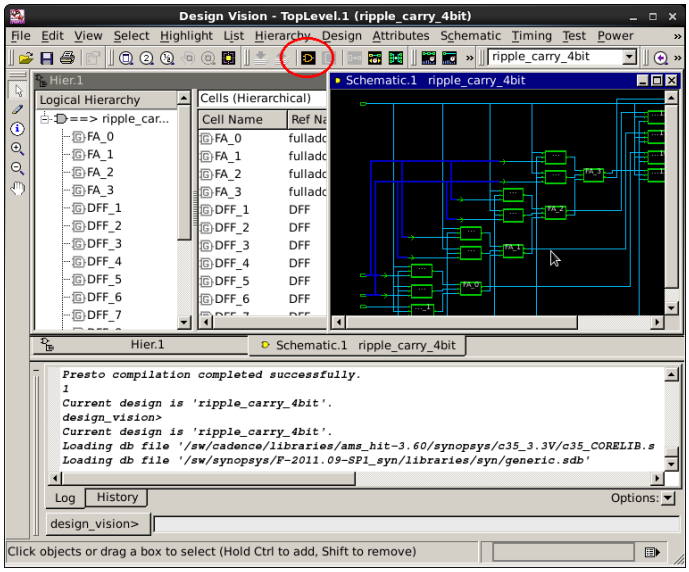


Figure 4: The main window after elaborate

5. In the graphical display, double-click on the clk input pin after zooming in. In the attributes menu, select Specify Clock. In the window that appears, specify the clock period to be 4 ns as shown in Fig. 5.

6. We shall use the command line interface of the Design Compiler for further steps. To map the design to the standard cell library, use the command `compile`. This may take a few seconds.

   ```
   compile
   ```

## 5.2 Timing, area and power estimations

After the design is compiled, we can obtain information about timing, area and power consumption of the implementation. To get a report of the timing of the design use the command

```
report_timing
```

The result should look similar to Fig. 6. The values that you obtain may differ.

7. We see that we have a positive slack (time margin) which means that the design meets the requirement given by the specified clock frequency. What is the slack in your case? What is the propagation time?

   ---

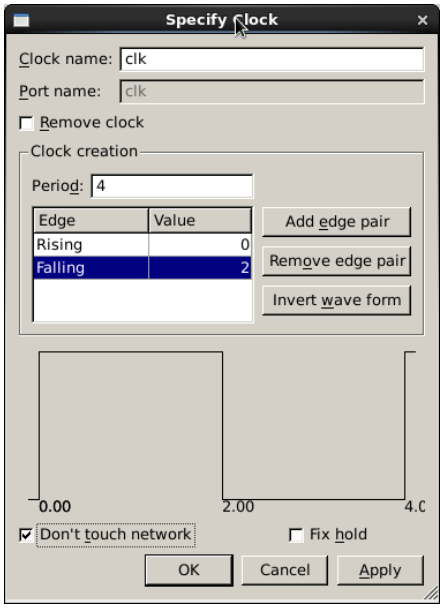


Figure 5: Clock frequency specification

8. To get a report of the area of the design use the command:

   `report_area`

   Here the unit is in $\mu m^2$. Give the total area in both in $\mu m^2$ and $mm^2$.

   ---

9. To get a report of the power consumption of the design use the command

   `report_power`

   Result?

   ---

10. With the following command, the power and toggle rate at different input nodes can be obtained:

    `report_power -net -include_input_nets`

    In the list, the toggle rate of the clock signal is seen as 0.5. Static probability is the probability that the signal is one. Toggle rate is the number of transitions during 1 ns. What are the default toggle rates of a, b and cin?

    ---

11. Using the toggle rates above, calculate the transition activities $\alpha_{clk}$,$\alpha_a$, $\alpha_b$ and $\alpha_{cin}$.

    ---

12. The reset signal should be constant at 0. This is done using the command:

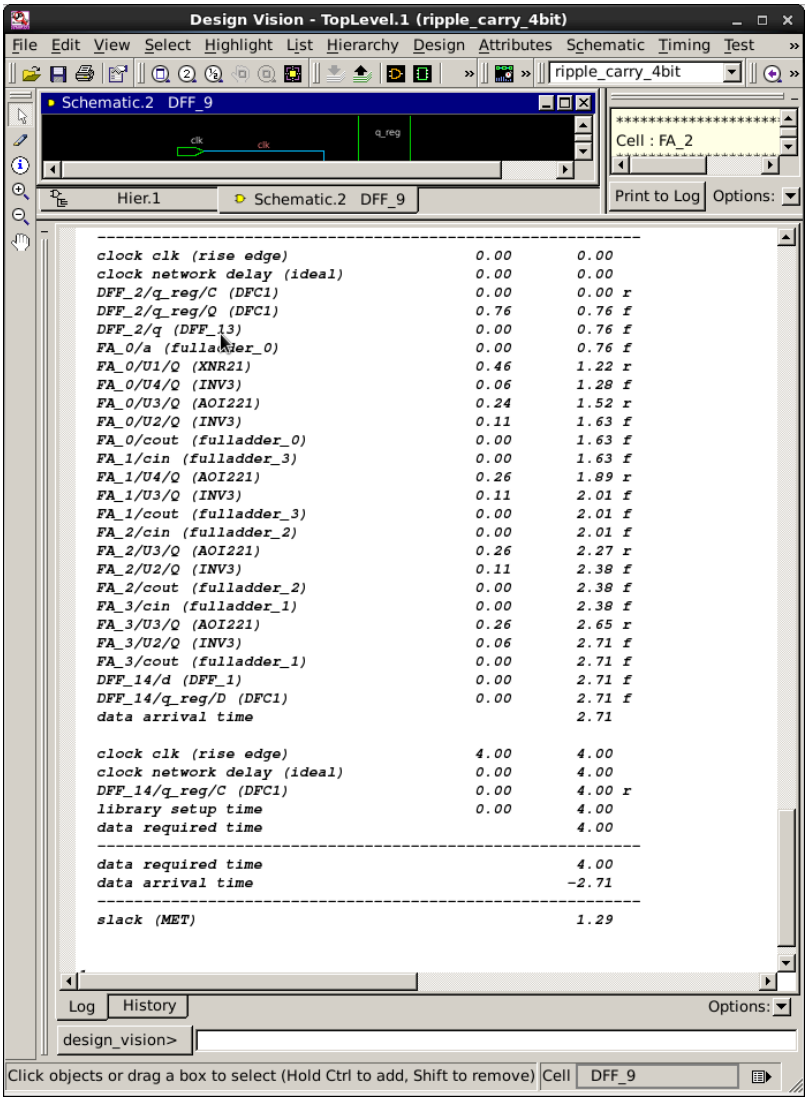


Figure 6: Timing report

```
set_switching_activity -static_probability 0 -toggle_rate 0 reset
```

The static probability and the toggle rate of the output signals of the registers must be specified by the user. Assume that a, b and cin are glitch-free and set the toggle rate for the output of all input registers.

```
set_switching_activity -toggle_rate 0.125 cin_pipe
set_switching_activity -toggle_rate 0.125 a_pipe
set_switching_activity -toggle_rate 0.125 b_pipe
```

Use the command `report_power -net` and verify that all the changes are done.
Use the command `report_power` to estimate the total dynamic power consumption of the circuit.
Results?

---

Use `report_power -hier`. How many percent of the power is consumed in the registers?

---

13. Now the compiled design needs to be saved as a verilog file which we later will use in Nanosim. Make sure that your top level block is selected.
    Select File / Save As ...
    Go up one directory and go down to the directory Nanosim.
    Choose File Format as Verilog and File Name as `ripple_carry_4bit.v`
    The switch "Save All Designs in Hierarchy" must be enabled.
    Save. Quit Design Analyzer by selecting File / Exit.

## 5.3 Information about some useful commands

The result from a power estimation can be presented in different ways. The first one makes a summary of the overall power consumption. The -cell switch presents the power consumption by each cell. Similarly, the -net switch presents power consumed in the nets of the design. Finally, the -hier switch will give you a more detailed presentation of the overall power consumption.

```
report_power
report_power -cell
report_power -net
report_power -hier
```

In addition to the power consumption, these commands also give information about how the consumption is estimated. For example, the command below returns information of the switching activity used when computing the power for each net.

```
report_power -net -include_input_nets
```

To obtain realistic power consumption estimations, the signal properties at different nodes in the design can be changed. The default the probability for an input signal to have the value 1 is 0.5. A default toggle rate will be derived from this value as well. Thus, when the probability for a one is changed, the toggle rate is changed as well. However, by assigning the toggle rate to a value, the probability of the signal being one is unaffected. The signal properties can be changed using the command `set_switching_activity`. For example, to change the probability of a value one on the reset signal to 0 use the command

```
set_switching_activity -static_probability 0 reset
```

In a similar way the toggle rate for a signal, in this case, the MSB of the input a, use

```
set_switching_activity -toggle_rate 0.1 a[3]
```

If the toggle rate of all elements in a vector a should be set,

```
set_switching_activity -toggle_rate 0.1 a
```

It is also possible to get the default values back for all signals using the command

```
reset_switching_activity
```

## 5.4 Further information

There is a help function available for the command you write on the command line. To get, for example, all possible switches for the `report_power` command, enter the following command on the command line.

```
help report_power
```

# 6 Estimating power using Nanosim

1. Make sure that the current directory in the terminal window is Nanosim. We will run Nanosim directly from the terminal window. Load the module:

   ```
   module load synopsys
   ```

2. Run nanosim:

   ```
   nanosim -nvlog ripple_carry_4bit.v -m ripple_carry_4bit -nspi spice.sp -L
   /coop/e/eks/course/TSTE85/c35_CORELIB -nvec test050.vec -c cfg -t 4008.00
   -o ripple_carry_4bit_050
   ```

   module load synopsys This command is also found in the file `run050` (It should be possible to run the file with: `sh run050` in the terminal window). The clock period is as before 4 ns and the data inputs (a, b and cin) are random. The switches of the command are explained below.

   ```
   -nvlog, the verilog-file to read,
   -m, the toplevel of the design,
   -nspi, the spice-file to read
   -L, the standard cell library that should be used,
   -nvec, the test vector file,
   -c, the configuration file,
   -t, simulation time in ns
   -o, output file name
   ```

   Look into the files to see how different parameters are specified. After the execution finishes, a report shall be displayed. You can ignore the warnings that says "...model already exists, ... is ignored". Power consumption?

   ---

3. Open emacs from the terminal window

   ```
   emacs &
   ```

   and take a look at the generated file `ripple_carry_4bit_050.cnt` (or use the command `cat` in the terminal window). How many transitions does the clk do?

   ---

4. Calculate the transition activity of the clock signal.

   ---

5. Approximately, what is the transition activity of all data inputs?

   ---

6. Give the number of transitions found in `cin, c_1, c_2, c_3` and `cout_pipe`.

   ---

   ---

7. Why is the number of transitions higher in `c_out_pipe` than in `cin`?

   ---

   ---

8. Consider the signal `sum_pipe[3]` and the output `sum[3]`. Number of transitions? How can the number of transitions in these two nodes differ as much as they do?

   ---

   ---

   ---

9. Now we run a simulation where the inputs have half the activity of the previous simulation (run025).

   ```
   nanosim -nvlog ripple_carry_4bit.v -m ripple_carry_4bit -nspi spice.sp -L
   /coop/e/eks/course/TSTE85/c35_CORELIB -nvec test025.vec -c cfg -t 4008.00
   -o ripple_carry_4bit_025
   ```

   Power consumption?

   ---

10. Now, the transition activity of the inputs have been reduced to half, but why is the power consumption not reduced to half of the original?

    ---

    ---

    ---

Congratulations! You have finished the first lab in TSTE85. Show your results and answers to your laboratory assistant.