CHAPTER 1

Power Estimation: probabilistic techniques

1.1 Probability and Activity Calculation: Simple Logic Gates (TL & TP)

The probability of having a ‘1’ at the output of a logic function depends on the inputs probability and on the logic function itself. The activity of the output of a logic function can be difficult to estimate if no assumptions are given on the inputs statistics. However if inputs are uncorrelated in time (actual values do not depend on previous values) and indipendent among themselves (spatially uncorrelated), the computation is somewhat easy, especially for combinational circuits with few logic gates.

During your class you’ve seen how to calculate the output probability and activity of a few logic gates.

Suppose the inputs are uncorrelated and have $P(inputs = '1') = \frac{1}{2}$. Compute the output probability $P(output = '1')$ and the output activity $A(output)$ of the gates in figure 1.1.

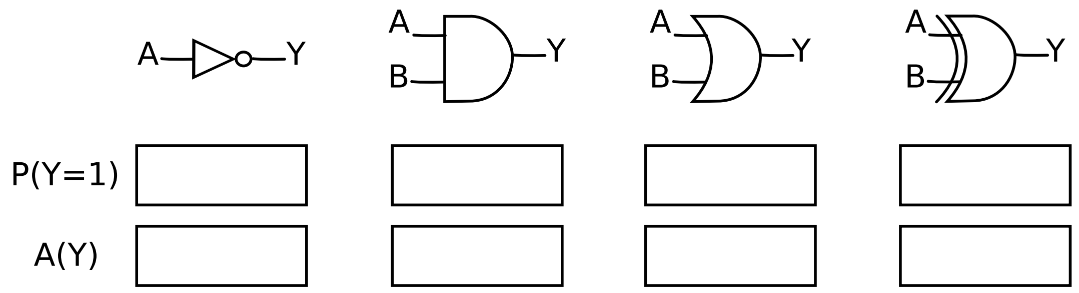


Figure 1.1:

The file **tb_prob.vhd** in directory

/home/repository/lowpower/ese1

contains the description of four logic gates:

INV, AND, OR and XOR

Their inputs are generated by a random number generator (LFSR, *Linear Feedback Shift Register*) which is contained in file **lfsr.vhd**, that you have to copy as well and compile before compiling and simulating **tb_prob.vhd**. An instance (ULFSR) of the component LFSR16 (16 bits LFSR) is instanciated.

Remember to copy all the required files within your home directory before starting the simulation or synthesis.

From the Simulate menu select *work, tbprob, test* with the a 100ps resolution[1]. Before running the simulation of the file **tb_prob.vhd** launch the simulator command

power add *

to be typed in the command shell. This command is used to compute the toggle count and probabilities of all signals (*) by means of simulations.
Now run the simulation as usual and look at input/output waveforms.
Now launch the command

power report

in the command shell. A report is typed out on the screen telling you, among other things, how many toggles every signal has experienced (Tc), the time the signal is at '1' or at '0'. You can save this report on a file by extending the previus command to

power report -file filename

From these values you should be able to estimate the signal probability and the activity. Use the toggle count of the clock Tc(clock) to evaluate the number of clock cycles (mind that the number of clock cycles is not the toggle count of the clock: $N_{ck} = \frac{T_c(clock)}{2}$).

Compare the estimations at various clock cycles[2] as indicated in the following table 1.1 with previous calculations.

Tc(CK)	Tc(INV)	Tc(AND)	Tc(OR)	Tc(XOR)
20				
200				
2000				
20000				
200000				

Table 1.1:

Remark point

How to obtain output probability and output activity of a circuit, output probability and output activity of the gates in figure 1.1, significance of table 1.1, how to obtain a power report output file.

1.2 Probability and Activity Calculation: Half and Full Adder (TL & TP)

Compute the output probability and activity of a **Half** and a **Full Adder** (in figure 1.2) as a function of the inputs probability starting from their truth tables.

[1]ATTENTION: this is the simulator resolution;: it must be lower than the minimum delay you are using inside the circuit. It is of utmost importance you carefully redefine it in each simulation you run

[2]Remember that you must restart the simulation each time you change the simulation lenght in order to avoid the superposition of the old simulation results on the new ones. For your comfortable working it is possible to create a simple script in your directory (with a common text editor) which includes the command sequence to be executed each time. E.g.:

```
add wave *
power add *
run xxx ns
power report
.....
```

To execute the script, in the modelsim command shell type: **do script-filename**.

	half adder	full adder
P(S=1)		
P(Co=1)		
A(s)		
A(Co)		

Figure 1.2:

Now compute the output probability and activity of a **Ripple Carry Adder** made of Full Adders as in figure 1.3 and for the special case where all inputs are equiprobable and uncorrelated.

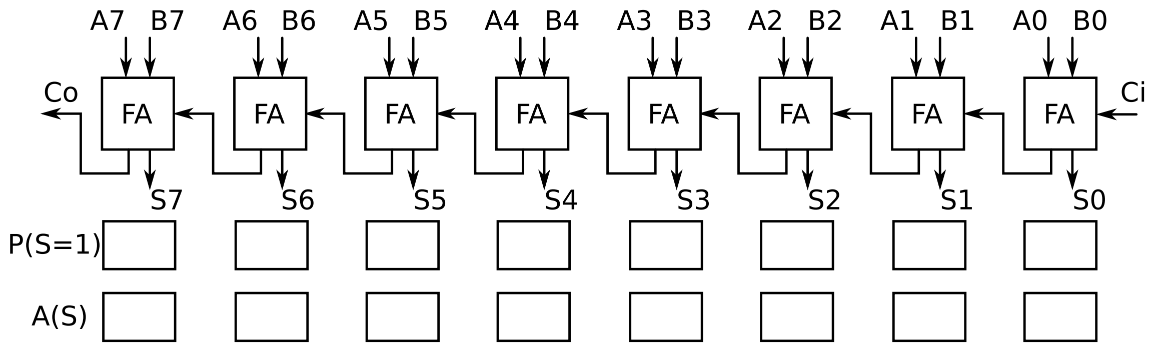


Figure 1.3:

Suppose the inputs are uncorrelated and have $P(A = '1') = 0.4$ and $P(B = '1') = 0.6$. What are the differences?

Now copy and simulate the files **fa.vhd**, **rca.vhd** and **tb_rca.vhd** (full adder, structural RCA and test bench). **Remember to choose the correct resolution (see the previous section).** Notice the **generic map** that is used in this case to assign two different delays, one for the sum **S** bit (**DRCAS**) and one for the carry **C** bit (**DRCAC**). Notice also that a LFSR is used again for the input random generation.

As for the previous exercise, compare your calculation with the simulation results using the command **power report**. Is there an agreement? It should.

Now add a delay to the carry computation so that it is equal to the sum one, e.g.

$$DRCAC = DRCAS = 0.025\,ns$$

(there is already another instance UADDER2 to uncomment). Compile and simulate again and compare with the previous results using **power report**. The results largely differ. What's happened?

Now use both instances of the adder with the two different delay assignment. Simulate for two hundred of cycles and compare the two waveforms. Expand the waveform for the array types of sum bits S1 and S2 by clicking the + sign. What is the difference?

Compute the total activity of the two adders

$$A(S) = \sum_{i=0}^{N-1} A(Si)$$

What is the overhead computation of the second adder?

Do you know what is the worst case activity? Load the file **tb_rca2.vhd** and try to understand what is going on.

Remark point

Output probability and activity of the Half and Full Adder; output probability of the RCA given uncorrelated and equiprobable inputs; check of the waveform simulation of RCA; power report output file; power report output file and understanding of the case DRCAC=0.025ns; computed activity for the two adders;

1.3 RCA synthesis and power analysis (TL & TP)

The aim in this section is to analyze how the power is computed after the synthesis phase by Power Compiler, the Synopsys engine included in the synthesizer able to analyze power and to optimize it if requested.

You should remember by previous lab the preliminary steps for synthesizing your gates:

Directory Generate a new directory for your synthesis:

```
prompt> mkdir syn
```

Copy in there files: **fa.vhd** and **RCA.vhd**

```
prompt> cp fa.vhd syn
prompt> cp rca.vhd syn
```

and file **.synopsys_dc.setup** you have in your setup directory:

```
prompt> cp ../setup/syn/.synopsys_dc.setup syn
```

Change files Now in the syn directory

```
prompt> cd syn
```

open both files and **comment the delay directives**, that is the generic declarations (in entity and components) and the 'after' declaration (timing information are unuseful for synthesis, as the used library gates have their own timing properties).

Call Design Vision In a shell in directory syn set the environmet variables:

```
prompt> setsynopsys
```

and launch the synthesizer:

```
prompt> design_vision &
```

Analyze, elaborate and compile In sequence, analyze (menu File→Analyze) both files in the correct hierarchical order (from bottom to top); elaborate (menu File→Elaborate) the RCA structural configuration in the WORK library; synthesize it (menu Design→Compile Design)

Clock We are about to analyze power: you know that power dissipation is related to the operating frequency, thus it is extremely important to remember that the correct clock period must be defined prior to analyze circuit power, otherwise a default clock period will be used.
In our case of course we don't have a real clock, but we can understand the minimum possible period by analyzing the critical path: in the design-vision-xg-t shell (bottom line of the GUI) write the command:

```
design-vision-xg-t> report_timing
```

you shoud obtain around 0.78ns thus you can decide for a 1n clock period:

```
design-vision-xg-t> create_clock -name clk -period 1
```

such clock “clk” is not a real signal, but is used for power computation.

Power The simpler report on power can be obtained by typing:

```
design-vision-xg-t> report_power
```

You can save a report by simply redirecting the display on a file:

```
design-vision-xg-t> report_power > rca-pow-report
```

(hereinafter you are in charge to decide if and when to save reports). Now we want to understand the origin of such contributions using more detailed report options.

```
design-vision-xg-t> report_power -hier
```

In this report in the first row the various power contributions for the top cell RCA are detailed again, while in the other rows the same contributions are detailed for the six full-adders. Power dissipations are similar except for the FAI_8: why?

It is interesting to detail more in depht now the origin of such power contributions. It is useful then to enter one of the FA by typing:

```
design-vision-xg-t> current_instance FAI_1
```

Now we are working on this full adder only. Display and save the global FA contributions:

```
design-vision-xg-t> report_power
```

and remember the three contributions. Now let’s analyze their origin:

```
design-vision-xg-t> report_power -cell
```

again you can see the details on the four contributions for each of the internal instances U1, U2, U3, U4: the sum of each type of contributions on the three instances is clearly correspondent to the total value found in previous report and to the correspondent row in the hierarchical RCA power analysis.

The only remaining point is to understand how the net switching power is estimated, as the internal and leakage are dependent on the gate only and are not related to the circuit. The interesting report from this point of view is:

```
design-vision-xg-t> report_power -net -verbose
```

In the first part of the report you have the capacitance associated to each node of this circuit, the static probability and the correspondent switching power dissipation. Why do you think that net S has a 0 dynamic power contribution? Can you found a correspondence between the switching activity detailed here and the values you found in previous exercise on RCA?

Finally let’s go up to the RCA level:

```
design-vision-xg-t> current_instance
```

and request a net power analysis at this higher level:

```
design-vision-xg-t> report_power -net -verbose
```

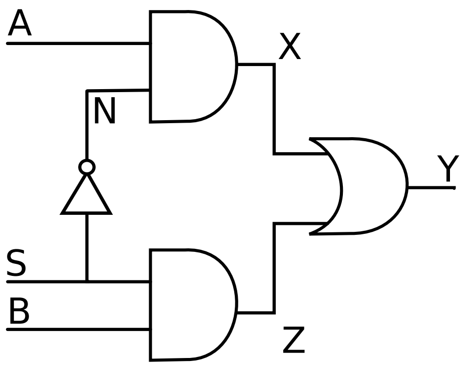
Can you understand the contributions and to analyze their correct correspondance to previous contributions?

Remark point

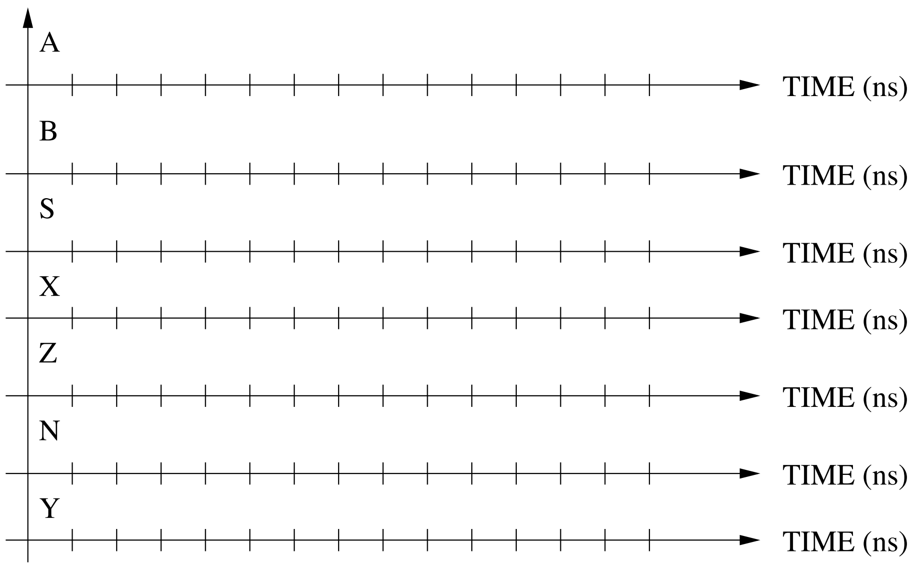
Power reports and contributions analysis.

1.4 A simple MUX: glitch generation and propagation (TL Homework, TP optional)

Copy now the file **tb_mux21_glitch.vhd** into a *vhdlsim* folder. Open it and identify the processes. Do they implement the following scheme?



Notice that the inversion takes 0.1 ns of delay to be completed. Now simulate this new VHDL code and observe the waveforms. Can you justify the behavior of the output waveform? Where does the glitch come from? Sketch the waveform of the nodes X, Z and Y considering the input variation used in the test bench. Is there any other inputs variations causing such an output behaviour?



Note: an undesired glitch means that two undesired transitions occurred. As a result, the node capacitance associated with that node will be switched once high and once low, then wasting energy without computing. What is the energy wasted during this transition (suppose C the node capacitance and Vdd the power supply)? Write the formula in the space below.

Remark point

Plot of all the gate signals explaining the glitch generation, mux VHDL netlist, mux VHDL test bench, signal waveforms, the formula for the vasted energy due to the transition.

1.5 Probability and Activity Calculation: Syncronous Counter (TL homework, TP optional)

Consider the structure in figure 1.4. Suppose that the DFF is initially zeroed: try to figure out what happens at nodes A0, Sum, S and Cout for a few clock periods. What happens if B0 goes to zero? Which is the expected activity at node S with respect to the clock activity?

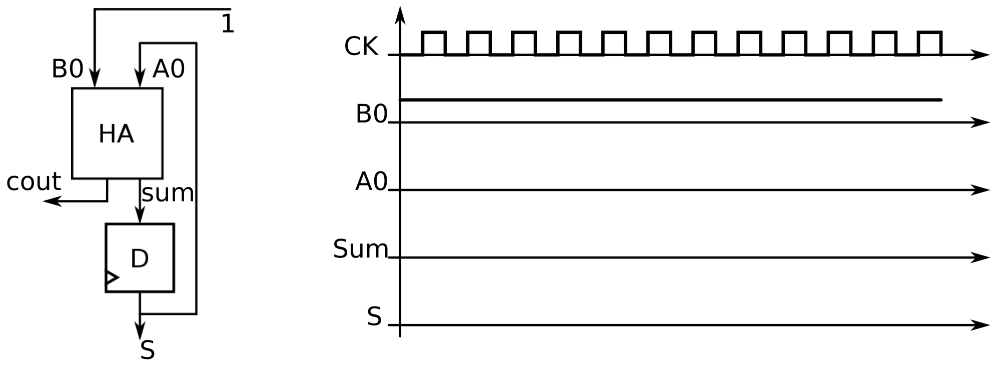


Figure 1.4:

Now suppose to build the structure in figure 1.5 in which the carry out of HA_i becomes one of the addendum of HA_{i+1}. What does the structure rapresent? Try to plot as you did before the output not only for S0 but for S1 and for S2 as well.

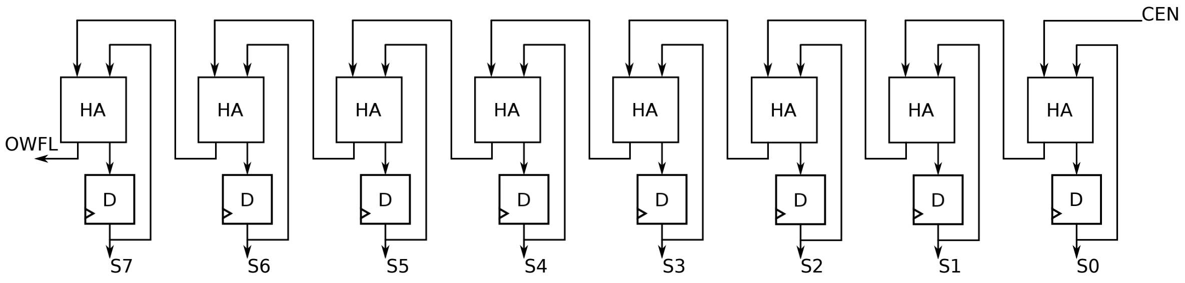


Figure 1.5:

You should find that this is a syncronous counter, counting from 00000000 to 11111111. It should be easy for you to count how many transitions will occur at the outputs, from S0 to S7, if a complete count from 00000000 to 11111111 goes on. Fill table 1.2 with the expected values.

Note that signal **CEN** behaves as a count enable, that is, the counter starts or increments the output values, only if CEN is '1'. What does rapresent the output OWFL, that is the carry out of the last HA? Which is the activity you expect for it?

Now copy from the usual directory the VHDL files **ha.vhd, fd.vhd, counter.vhd, tb_counter.vhd** describing the structure commented above into a *vhdlsim* folder. Simulate the counter and run the simulation for about 260 clock periods (note that the CEN is '1' only after a few clock periods; note

Signal	Number of Transitions
Clock	
S0	
S1	
S2	
S3	
S4	
S5	
S6	
S7	

Table 1.2:

also that the clock period is defined in the test bench). Before running the simulation (and adding waveforms in the usual way) type the command:

power add /testcount/UCOUNTER1/*

so that you will have the power report for the signals inside the counter instance.
After you run the simulation check if the output is correctly counting, and zoom around the 128-130 clock periods: note that the OWFL is active. Do you expect that the OWFL signal remains always zero before this moment? Does this expectation agree with the simulation results?

Now type **power report**: compare the transition results you are reading with the values you computed in previous step and wrote in table 1.2.
Consider now the output of the HAs (not syncronous): is the transition number coherent with the ones of the DFFs? Which is the activity of the carry out vector? What is happening inside the counter? Remember what was happening with the Ripple Carry Adder... and in the MUX before and try to understand if it is possible to change something in the VHDL code.

Now try to change the clock period from 2ns to 0.8ns. What do you think the output will be? Simulate for 30 ns and analyze the counter output, in particular around 14-16 ns; does the counter count correctly?

Remark point

Behavior of the structure in figure 1.4; number of transition for the output: comparison between the expected values and the values reported by power report; power report file; comments on the counter internal activity; comments on the increased frequency results.