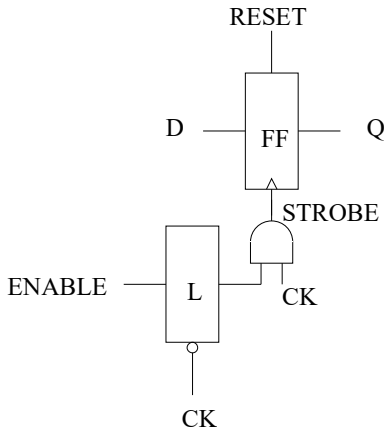
CHAPTER 3

Clock gating, pipelining and parallelizing

In this chapter you will experience some techniques for reducing power consumption by optimizing the circuit architecture. From directory /home/mg.lowpower/lowpower/ese3/ copy all the VHDL files: cp /home/mg.lowpower/lowpower/ese3/* .

3.1 A first approach to clock gating (TL & TP)

In **ckgating.vhd** you find an implementation of the *clock gating* technique reported also in figure for sake of clarity. Notice the latch before the *and* gate. It is used to filter away glitches when the clock is high.



In the following exercise you will understand how the VHDL simulator works and how to avoid troubles both in simulation and design.
Copy and open the file **ckgbug.vhd**. Look at the signal declaration and in particular to the **std_logic_vector** type. It is an *array* of **BYTE** std_logic, where BYTE is defined as a "constant". Signals D1 and D3 are defined **std_logic_vector(7 downto 0)** which means that the Most Significant Bit (MSB) is in position 7 and the Least Significant Bit (LSB) is in position 0. Differently D2's MSB is in 0 and LSB is in 7.

Remember the **generate** statement used for multiple instantiations together with statement **for**. Byte=8 instances of the flipflops are used to create first register L1 and second register L2. L1's input is array D1 and its output is D2. D2 is on its turn input to L2 whose output is D3.
L2 is gated (STROBE signal in the Port Map) while L1 is not (CK signal in the Port Map). So D2 always copies D1 (with bit positions inverted!) while D3 copies D2 only when ENABLE='1'.

Suppose the reset has zeroed all values in register and try to mentally simulate what happens if input D1="01111111" and ENABLE='1'. Now simulate the VHDL code and compare your prediction to what the simulator says.

The behavior differs from what we expected. Why? It seems that D3 directly copies D1 on the clock rising edge. This is what really happens. In fact the clock gated (STROBE) arrives at the second register a "simulation step" later than at the first register. This happens because the simulator schedules the computation of the AND output after the CLOCK assignment (causality!). As a result all things goes like the AND has an internal delay. Therefore FIRST D2 changes and copies D1, then D3 copies D2 that already changed! This is a mistake we should avoid during simulation, but it is something that could really happen, if the clock gating circuit adds too much delay such that the HOLD TIME of the FLIPFLOP is no more respected.

How can we avoid this problem? A simple solutions consists in adding a Clock-to-Output delay. For example add 0.1 ps to the Q assignment in *fd.vhd*. Simulate again and see what happens. It should work now. But what if the *and* has its own delay of, for instance, 0.2 ps, that is larger then the Ck-Q delay? Apply this change, and try.

Remark point

Concept of simulation step and causality

3.2 Clock gating for a complex circuit (TL & TP)

The goal is: let's consider a circuit, and understand how it works, then synthesize it and report its power consumption. Then let's try and apply clock gating where possible and synthesize it again, report the new power consumption: is the total dissipation decreasing?

The circuit you are going to analyze implements the operations sketched in figure 3.1.

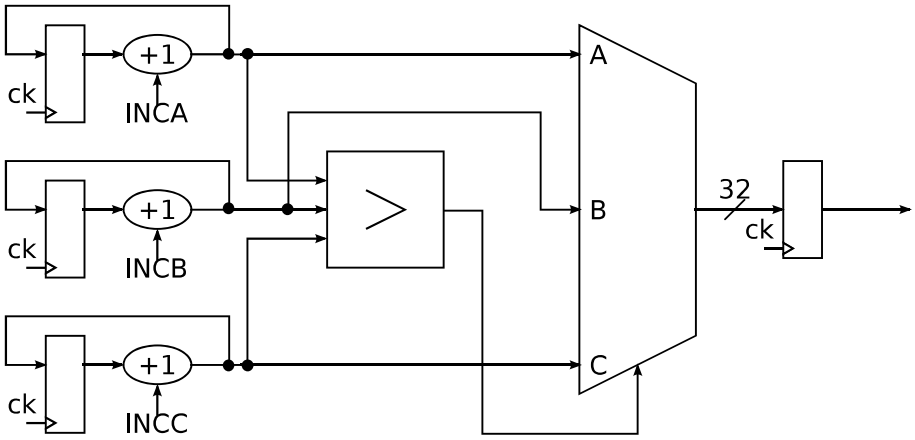


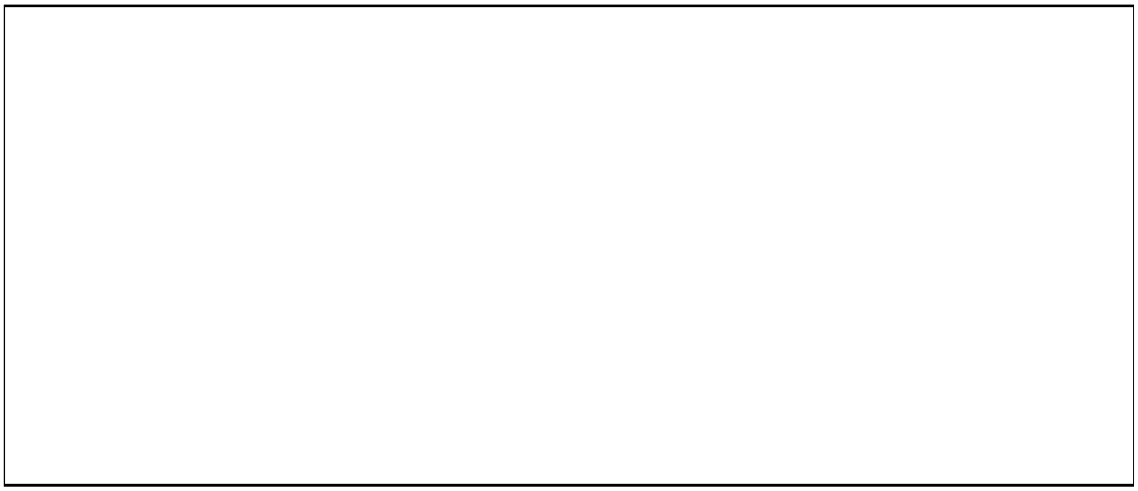
Figure 3.1:

You are also provided with the VHDL, that you will find in

/home/mg.lowpower/lowpower/ese3/inccomp.vhd

In the same directory you will find the test bench *tb_inccomp.vhd*. What does the circuit do? Now simulate it Are the results consistent with what you expected?

Now try to VIRTUALLY (pen and paper) apply the clock gating technique to the circuit: change the sketch in figure 3.1 to this purpose and draw it in the box below. Suggestion: use input signals INCA , INCB and INCC!
Note that you don't need to write the vhdl of this part, as the given vhdl is already written so that tha synthesizer is capable of applying clock gating if requested.



Now you are ready to **synthesize** the circuit. Use the directives in chapter 2 to initialize the synthesizer. Remember that the synthesizer wants a subdirectory *work* as well as the simulator, and it is better if you perform simulation and synthesis in two different subdirectories (just create a subdirectory **synth** and in it a subdirectory **work**, copy in **synth** the Synopsys setup file (.synopsys_dc.setup) and the VHDL files to be synthesized and launch **design_vision**).
You should perform a first synthesis without considering clock gating. After you launched **design_vision** go to the bottom **Command Line (Log)**. Hereinafter you will be given instructions to be typed in the command line (should you insist in using menus and windows, well, it's up to you ...). If you want to become a design_compiler-pro you can also try to write directly a script using as a reference the one you saved in the last lab and the hints you will find herein.

To read the file, type in the command line:

analyze -format vhdl inccomp.vhd

elaborate inccomp -library work -architecture behavioral

now synthesize it by typing

uniquify

compile -exact_map

You can navigate through the synthesized schematic. Now create a clock and report the power consumption:

create_clock -name ck -period 4 {ck}

report_power -include_input_nets

The tool creates a clock of period 4 ns (nanoseconds is the default time unit and is defined in the library of logic gates used for the synthesis). Pay attention hereinafter: he report power command

uses the clock for computing the power dissipation. So you always must check the clock period you are using when comparing different report power results. The option "-include_input_nets" forces the tool to consider the input nets for power calculation as well. Without this option the tool shows a power consumption figure accounting for internal nets and cells only.

To have more detailed information on power consumption you can perform a power report on nets by typing

report_power -net -include_input_nets

you will note that for *clock, reset* and inputs a static probability $p(res =' 1') = 0.5$ has been assigned, which is not correct. Let's understand how the rate is computed, so that we are able to change it.
The toggle rate shown in the report is a floating point value which accounts for the number of transitions of a net within a clock period. It can be specified differently according to the "set_switching_activity" command (see the manual page for more information). When the power report is printed out, the toggle rate is divided by the reference clock period: For example, if toggle rate is 0.5 in a determined net with a clock period of 5, the power report prints out 0.1. In formulas, if static probability $p1 = 0.5$, this is given by $2 \cdot p1 \cdot (1 - p1)/T_{ck} = 2 \cdot 0.5 \cdot 0.5/4 = 0.1$. However, under the Synopsys shell it is necessary to specify the toggle rate as the number of toggles within a clock period rather than the product of two probabilities. This parameter, assigned automatically by the tool, is not correct both for the *reset* and for the *clock* signal, so correct it by typing

set_switching_activity -static_probability 0.5 -toggle_rate 2 -clock ck {ck}
set_switching_activity -static_probability 0 -clock ck {rst}

where "-clock" is used to let the tool calculate toggle rate with respect to the clock named "ck". A toggle rate of 2 for the clock means that the signal in parentheses {ck} changes twice from 0 to 1 and viceversa within a clock period. Check if the toggle rate for the clock signal is correct by running the power report for all the nets in the current design with

report_power -include_input_nets

Did something change? Finally impose a lower input probability to inputs as well

set_switching_activity -static_probability 0.12 -toggle_rate 0.025 -clock ck {INCA INCB INCC}

and analyze the power report. In this case the obtained power report is for sure more accurate because it has been calculated by assuming realistic input probabilities (save it).
You can also annotate the number of gates used for this synthesis by typing:

report_cell

You are now ready to apply the clock gating technique. Luckily it will be performed for you by the synthesizer if you will give him the right directives. So reset previous results by selecting on the menu **File→Remove all Design**
and tell the synthesizer you want a clock gated synthesis by typing

set_clock_gating_style

a few more instructions are needed before compiling

analyze -format vhdl inccomp.vhd

elaborate inccomp -library work -gate_clock

propagate_constraints -gate_clock

uniquify

Remember that if you want to understand the meaning of a command, let's say *commandname*, you can type

man commandname

a little bit of hacking could help, now and then ...
You can synthesize now:

compile -exact_map

create the clock

create_clock -name ck -period 4 {ck}

and analyze the power report:

report_power -include_input_nets

As before you should change the probability of *reset*, *clock* and inputs and see how the power change

set_switching_activity -static_probability 0.5 -toggle_rate 2 -clock ck {ck}
set_switching_activity -static_probability 0 -clock ck {rst}
set_switching_activity -static_probability 0.12 -toggle_rate 0.025 -clock ck {INCA INCB INCC}

Then analyze the power consumption variation with respect to the previous case

report_power -include_input_nets

and analyze the power report [1]. Did you note a power variation with respect to the previous synthesis? Report the gate number **report_cell**: is it increased or reduced?
Now it should be instructive to understand what's happened in this last operation: try to navigate the schematic and find a clock gating element. Is something similar to what you expected?

Remark point

How to apply clock gating, how to annotate activity.

3.2.1 Some more clock gating? (TL & TP)

Now a question: are we sure that we applied clock gating wherever possible? The VHDL you synthesized imposing the clock gating style helps in inserting the clock gating elements for sure the three initial registers, but the register to the output pin is not gated: why? Something could be inserted in the original VHDL to help the synthesizer...

Once you have changed the VHDL code synthesize it as before without clock gating and report the power consumption (three steps: no probability change, change the reset, change the input). Check

[1] If you're getting bored check the history window or type **history** in the command window: in both cases a list of the last commands you typed is displayed and a sequential number is associated to them, let's say *nb*. If you type in the command window **!nb** the associated instruction is automagically executed...

the number of gates as well.
Finally perform again the clock gating synthesis as before, analyze the power consumption (three steps) and the number of gates. Navigate the schematic and check where other clock gating block has been inserted.

Remark point

How to force the clock gating using VHDL.

3.2.2 An automatic way to annotate activities (TL & TP)

In the previous exercise you have seen that a wrong activity annotation can lead to wrong results in the power estimation. It is strictly necessary to put a reasonable activity on the nodes, also in the cases that circuits are very complex. In such cases it is not practical to annotate the activity of each node with the command "set_switching_activity" and it is necessary to make the annotation process automatic by using standard files called SAIF files.

Let's consider the overall annotation process in figure 3.2: The process is inclusive of two parts, one in the VHDL simulator and another in the design compiler (Synopsys synthesizer). First (1), after having analyzed the design, the synthesizer can generate a forward annotation SAIF file which accounts for the names of the netlists on the design. It is like a SPICE netlist file but at this step the tool leaves the "activity" fields empty for each node. Once the VHDL simulator has read the forward file, after having run simulation (2), it updates this file by writing the activities in each node[2] and finally create the backward SAIF file (3). The synthesizer after compilation can read the backward annotation file and update the properties of the nodes (4) with the activities calculated from the VHDL testbench.

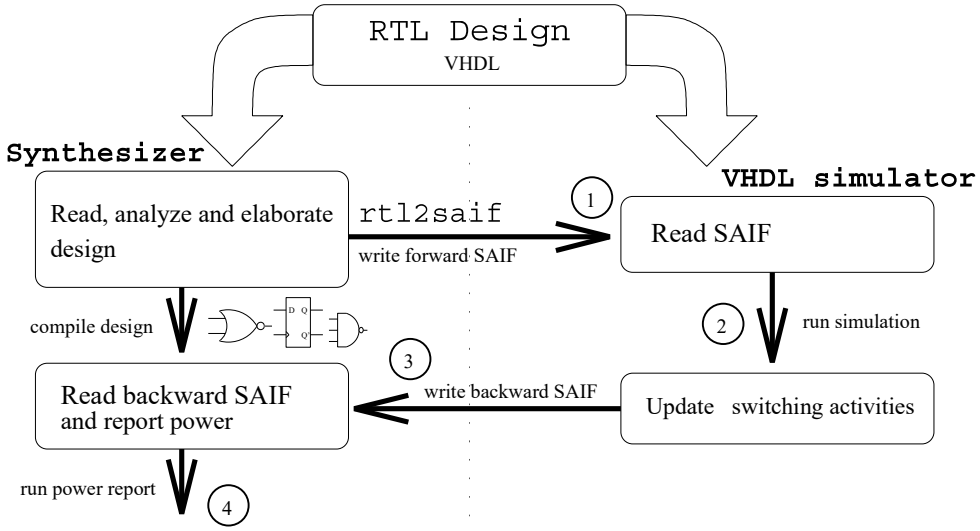


Figure 3.2:

We will now see how to realize this backannotation process. Considering that you will use both Modelsim and Synopsys, for convenience create a new folder in your *ese3* directory called `saiftest`. Inside this directory create two folders called `modelsim` and `synth`. By using two different command shells you will run both Modelsim and Synopsys inside `modelsim` and `synth`, respectively. Don't

[2]In this case the concept of node corresponds to the concept of "signal" in the VHDL language, thus variables are not mapped as nets and not annotated

forget to copy the `.synopsys_dc_setup` file in `synth`. Copy the source VHDL files `inccomp.vhd` and `tb_inccomp.vhd` inside `saiftest`.

To run SAIF annotation under Modelsim it is necessary to include a Synopsys shared object file in the library list (so that the two tool can talk together). Copy then the Synopsys library **/home-/mg.lowpower/lowpower/ese3/so/dpfli.so** in your local directory `saiftest/modelsim`.
During thesimulation you are going to perform, the inclusion of the library can be specified with the command `vsim` by adding the additional library among the command parameters. However, to make the execution of this laboratory session easier, part of the commands will be executed by using two scripts called `forward.do` and `backward.do`. Copy the file `forward.do` from **/home-/mg.lowpower/lowpower/ese3/do/forward.do** in `saiftest/mentor` and the file `backward.do` from the same location in `saiftest/synth`. Now you have a folder called `saiftest` including the VHDL files and two folders called `mentor` and `synth` which include one script each. In addition, in the `mentor` directory you have a ".so" file. Hopefully now you are ready to start.

For this example let's refer to the `inccomp` entity included in the file `inccomp.vhd`. Run synopsys[3] and after having analyzed and elaborated `inccomp.vhd` type

link
rtl2saif -output ../power_rtl_forw.saif

where "power_rtl_forw.saif" is the forward annotation file (just take a second to look at it). The link command is used to resolve the references of the internal sub-blocks in the design if any.
Now, without closing Synopsys, run Modelsim from another shell working in the `modelsim` directory (remember the `vlib work` command). After the design compilation (that is the two files `inccomp.vhd` and `tb_inccomp.vhd`), run the script `forward.do`. It reads the forward annotation file `power_rtl_forw.saif`, simulates the testbench and updates the activities in the SAIF file by writing the backward annotation file (named **backward.saif**). Open the file and check what's inside the script: This could be useful to understand how you can adapt it to other designs.

Now go back to Synopsys and run the script `backward.do`. It reads the backannotated file and updates nets information. Check what's inside the script and the power consumption of the design (with the command **report_power -include_input_nets -net**). The activities should be annotated from the Modelsim simulation.

Note: This automatic process is inherently based on the VHDL testbench, thus on the generation of an at most "complete" set of inputs to the design. Whether simulation conditions in the VHDL testbench are not "realistic", the power consumption figure of merit can be completely wrong.

Remark point

How to annotate activity automatically by using SAIF mechanism.

3.3 Pipelining and parallelizing (TL Homework, TP optional but STRONGLY suggested)

Now we will see how to greatly reduce the power dissipation by exploiting the parallelization and/or pipelining of computation that allow reducing the power supply voltage and so save energy.

Let's consider again the previous datapath in figure 3.1. Suppose that all blocks have been characterized at 1 V, f=5 MHz in terms of delay, power and area as in the following table:

[3] Don't forget to set the environmental variables with `setsynopsys` or `setmentor` before running `design_vision` or `vsim`!

cell type	delay (ns)	power (μW @1 V, 5 MHz)	area (μm^2)
REGISTER	2.0 (ck→Q)	0.6	319.0
INCREMENT	40.0	2.55	256.0
COMPARATOR	84.0	2.16	161.0
MUX	14.0	1.67	117.0

Suppose also that delays follow this law with Vdd

$$T(Vdd) = k\frac{1}{1-\frac{Vt}{Vdd}} = k\frac{1}{1-\frac{Vt}{Vdd_{NOM}}\frac{Vdd_{NOM}}{Vdd}}$$

where k is a constant and Vt is the threshold voltage of CMOS transistors (supposed symmetric for NMOS and PMOS transistors). Normally the modern devices are designed to work at $Vdd_{NOM} = 4Vt$. Therefore if we introduce the normalized potential $u = Vdd/Vdd_{NOM}$ and use threshold voltage as 0.25 times the nominal Vdd, we can express delays as follows

$$T(u) = T(Vdd = Vdd_{NOM}) \cdot \frac{0.75u}{u - 0.25}$$

In an analogous way, the power can be expressed as

$$P(u) = P_{NOM} \cdot u^2$$

In the following figure normalized delay and power are reported.

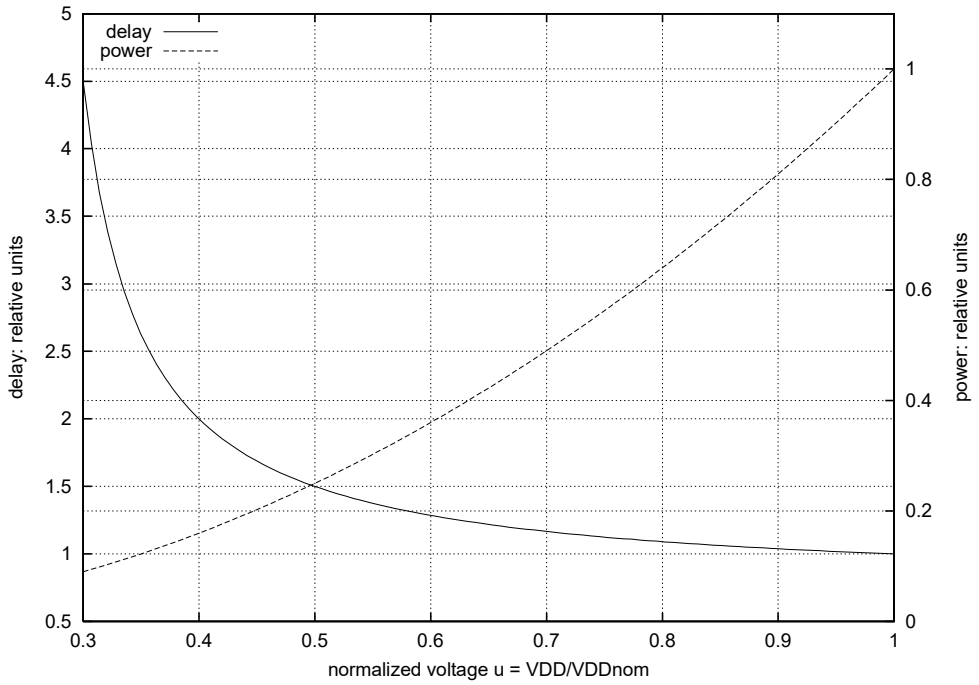
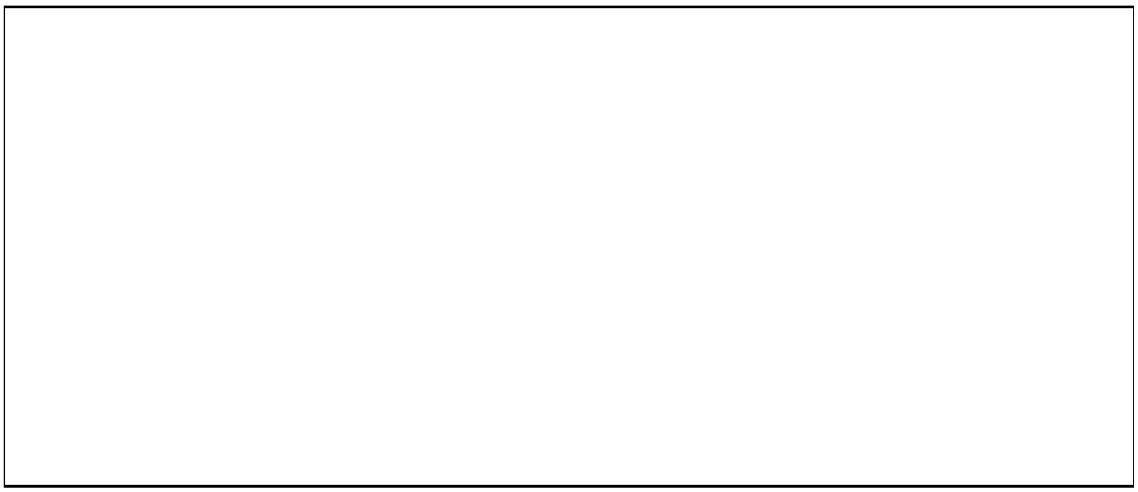


Figure 3.3:

Using the previous delay and power table, compute the allowed clock frequency and related power dissipation (suggestion: setup and hold time of registers is assumed negligible, power have to be recomputed at the new frequency of operation):

original solution	
delay (critical path)	ns
allowed clock frequency	MHz
power	μW
area	μm^2

Suppose now you want to utilize some parallelization in order to reduce the power dissipation at the same throughput (i.e. same frequency). You are allowed to duplicate the datapath, then almost doubling the switched capacitance, **but** you can work at half the clock frequency. Can you draw the new circuit?



Compute the new voltage supply, power dissipation and area, including the overhead of the output multiplexer you should have added.

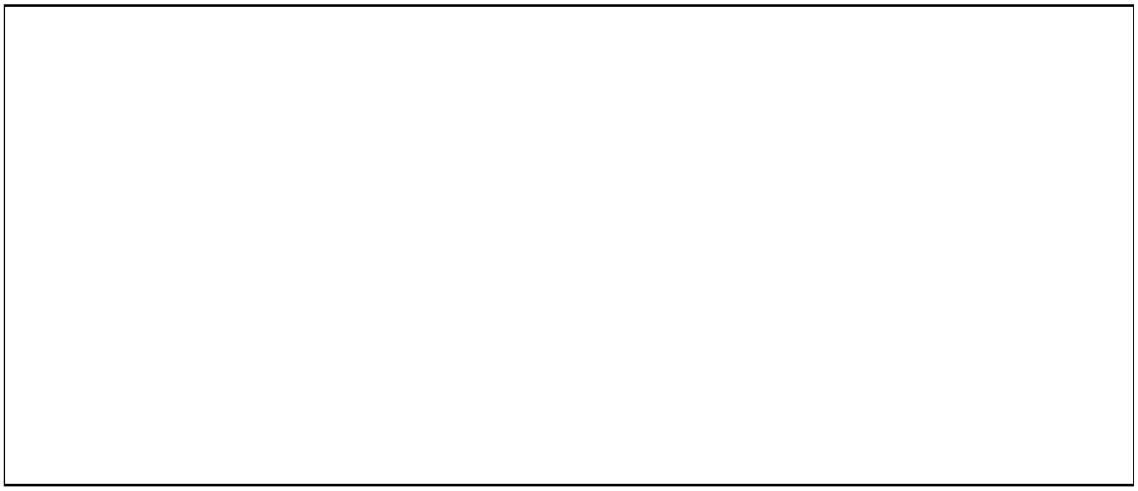
parallel solution	
delay (critical path)	ns
allowed clock frequency	MHz
power	μW
area	μm^2

Suppose now you want to employ pipelining, choose where to insert pipeline registers. Draw again the circuit.

Then compute the new supply, power and area figures taking also into account the overhead of new pipeline registers.

pipelined solution	
delay (critical path)	ns
allowed clock frequency	MHz
power	μW
area	μm^2

Try now to use combinations of both techniques, finer grain pipelines (new registers), and more parallelization degrees (e.g. triplication) and discuss advantages and drawbacks of these solutions and of the previous ones and compare them trying to carry out your own conclusions.

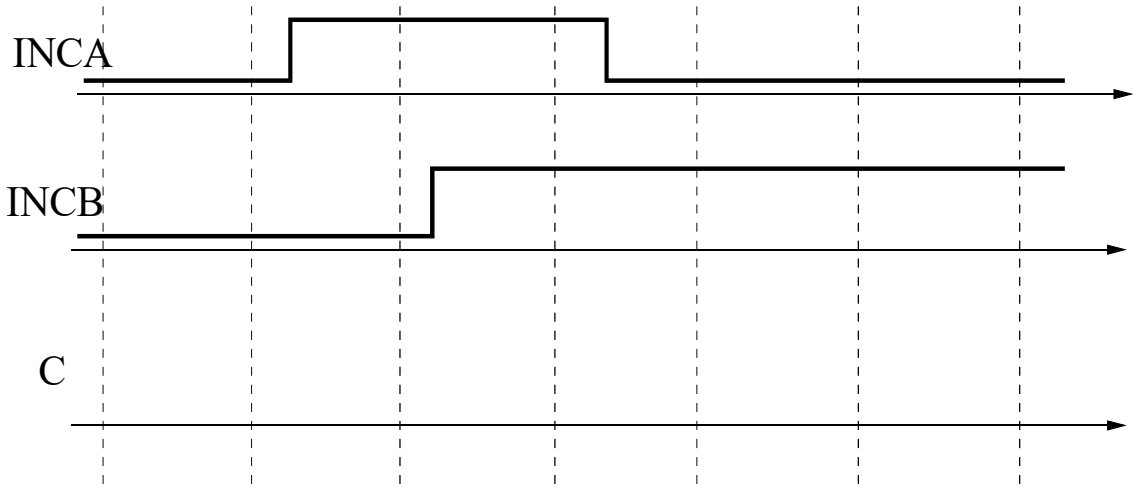


Remark point

How to modify an architecture accounting for parallel and pipelined low-power techniques, how to compute the power advantages.

3.3.1 Are you sure it was correct?

Look at the waveform in figure 3.3.1. Use such signals for finding the output C in the case of the original architecture. Then consider the parallel architecture and find the output C. Are the results coherent?



Remark point

Applying parallelizing and pipeline so that data coherence is assured.