UE-VLSI2 TP : Cadence RTL Compiler
Kévin Mambu, Nicolas Phan

Kevin Mambu

January 3, 2019

1 Set the environment

```
#!/bin/sh
export LM_LICENSE_FILE=30000@licences.cnfm.fr
export PATH=$PATH:/users/soft/opus/Linux/RC-13.12-000/bin
```

2 Load the libraries

n.b: all the libraries are located at `/users/enseig/tuna/ue-vlsi2/techno`.

Two types of cell libraries are available: `_Best.lib` and `_Worst.lib`. The difference between these libraries lies in the timing arcs of their cells : while the Best Library contains Best-Case timings, the Worst Library contains the Worst-Case equivalents. Each timing arcs are estimated via different PVT parameters (**Process - Voltage - Temperature**). These parameters can be found in the headers of each library.

```
/*****************************************************
 Synopsys Technology File

 genstf version 6.4
...
Process values given:
  Library nominal:                1.2
  Tech-file best case:            1.2
  Tech-file centre:               1.2
  Tech-file worst case:           1.2
Voltage values given:
  Library nominal:                1.08
  Tech-file best case:            1.20
  Tech-file centre:               1.08
  Tech-file worst case:           1.0
Temperature values given:
  Library nominal:                85
  Tech-file best case:            25
  Tech-file centre:               125
  Tech-file worst case:           125

  *****************************************************/
```

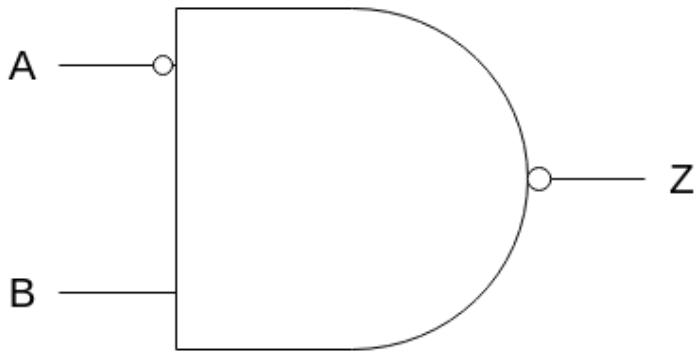
PVT specifications for cmos_120nm_core_Worst

```
/*****************************************************
 Synopsys Technology File

 genstf version 6.4
...
Process values given:
  Library nominal:                0.8
  Tech-file best case:            0.8
  Tech-file centre:               0.8
  Tech-file worst case:           0.8
Voltage values given:
  Library nominal:                1.32
  Tech-file best case:            1.32
  Tech-file centre:               1.32
  Tech-file worst case:           1.20
Temperature values given:
  Library nominal:                0
  Tech-file best case:            -40.0
  Tech-file centre:               0
  Tech-file worst case:           25.0

 *****************************************************/
```

PVT specifications for cmos_120nm_core_Best

In order to synthesize the RTL description, we will use cmos_120nm_core_Worst.lib. This library contains all the generic cells we will need, and with Worst-Case timings in order to thoroughly stress the longest combinational paths.

A practical example of difference in timing arcs between this library and its _Best equivalent can be seen when evaluating the ND2AHS cell, a "2 Input NAND w A Input Inverted and 1x Drive".



Gate-level schematic of the ND2AHS cell

3 Load the design

4 Elaborate

The elaboration process does not differ from the Synthesis per se. The elaboration step is the conversion of the RTL description to a model where every instance object (signal, process, arrays, etc) is converted into a numeric component (adder, multiplexer, register, etc). This step is part of the Synthesis process. Quote : "The first step, elaboration, is reading in your RTL file (which is text) and recognizing bits of code that represent real hardware structures. Once recognized, these are converted (in Vivado synthesis case) into "generic technology cells" - abstract things like registers, adders, comparators, multiplexers, arbitrarily wide gates, etc..."

So elaboration creates the netlist of **generic** technology cells.

The mapping process will consist in associating a cell from the chosen specific library to the generic cells. Example of RTL design before ekaboration/synthesis : ... After elaboration : ... After syntesis : ...

5 Check design

After looking at the report of the Design Check, it seems like there isn't any major warning. Still, we have notes regarding "Unloaded Sequential Pins", "Assigns" & "Constant Hierarchical Pins". None of these notes necessarily impact the functionnality of the design.

6 Synthesis

7 Reset

7.1 Synchronous & asynchronous resets

There are two reset signals in the design :

1. RESET_N is the external reset signal. Its whereabouts are not bound by any timing constraints and its value can change at any time, as the signal comes from outside the design through an interface pad. This is an asynchronous reset signal.

2. RESET_RX is the internal reset of the design. This one is synchronous as its value is refreshed at every rising edge of the clock.

First we should precise that while the activation of the reset ($0 \rightarrow 1$) is asynchronous, its deactivation ($1 \rightarrow 0$) has to be synchronized with the clock, because all the components in the design are sequential.

7.2 Hazards of meta-stability

Still the external reset is a potential hazard for the design, as it comes from the outside. We cannot simply assume that the propagation of the signal was flawless and must consider that it may have had issues on its way:

1. The external reset may come from a different clock domain. In that case, the signal would have been synchronous in its previous clock domain but would appear asynchronous when entering the design.

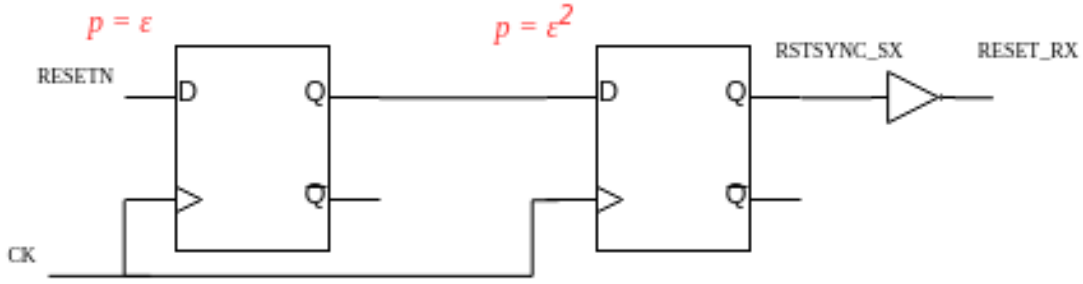
2. The external reset may be already asynchronous.

In any case, the external reset signal may arrive in the design without respecting constraints relative to the setup time of its imminent receiver (a flip-flop, for example). All these possibilities may bring the value of the signal to a metastable state. A case of meta-stability occurs when the signal is captured by a receiver during an edge, leading to an undetermined value being captured ($\frac{1}{2}$). In the case of a state register, for a Finite-State-Machine, the capture of a meta-stable value may lead the FSM to an undetermined state, ruining its functionnality. We have to prevent such a case from occuring.

7.3 Reset Synchronizers

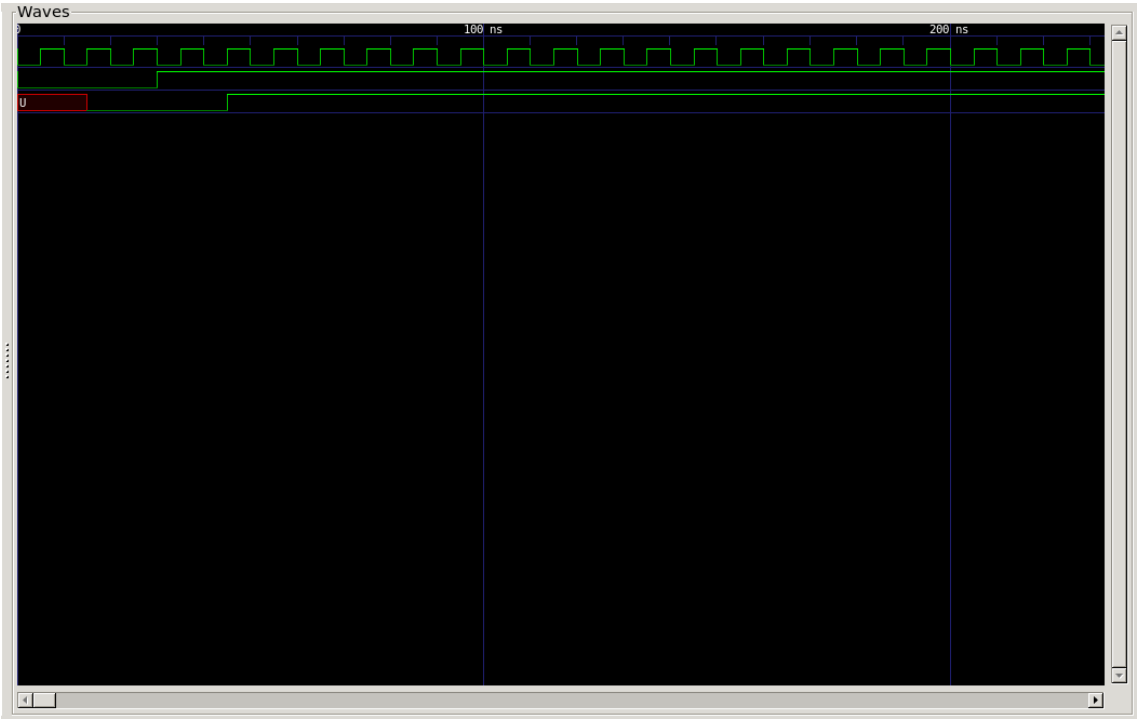
A Reset Synchronizer is a component which takes an asynchronous signal as input and outputs a synchronous signal. Every time the signal crosses a flip-flop of the Reset Synchronizer, its chances of still being meta-stable ddecreases.

Let ϵ be the probability for an asynchronous reset to bring a flip-flop to a meta-stable state. For n being the number of flip-flops on a given datapath, the probability of meta-stability being propagated through the datapath is ϵ^n.



Schematic of a 2-register Reset Synchronizer. In red are the corresponding probabilities of a propagation of meta-stability.

This model was implemented to the design. The verification of the functionnality was done using GHDL.



Waveform of the Reset Synchronizer using GTKWave

Listing 1: RTL description of the Reset Synchronizer

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity reset_synchronizer is
        port ( i   : in  std_logic;
               clk : in  std_logic;
               q   : out std_logic );
end reset_synchronizer;

architecture behav of reset_synchronizer is
        signal reg0 : std_logic;
        signal reg1 : std_logic;
begin
        q    <= reg1;
        clocked : process(clk)
        begin
                if rising_edge(clk) then
                        reg0 <= i    ;
                        reg1 <= reg0;
                else
                        reg0 <= reg0;
                        reg1 <= reg1;
                end if;
        end process clocked;
end behav;
```

8 Reporting

From the area report, the design is composed of 13878 cells, and has a surface of 220290 square cells.

The timing report gives the longest datapath of our design. We can see that its start point is the I_RI register and its end point the NEXTPC_RD register. The total delay of this datapath is 11.538ps. The maximum frequency, evaluated only from this timing report, equals $\frac{1}{11538 \times 10^{-12}}$ Hz, or approximately 86.670 GHz. The timing slack of a connection is the difference between its required time and its arrival time. In our design the timing slack is UNCONSTRAINED because no timing constraints have been specified yet. Because of this omission, no clock signal are emitted on the design.

When anayzing the lint timing report, we can see the following warning catergories :

1. Generated clocks without clock waveform
2. Inputs without clocked external delays
3. Outputs without clocked external delays
4. Inputs without external driver/transition
5. Outputs without external load

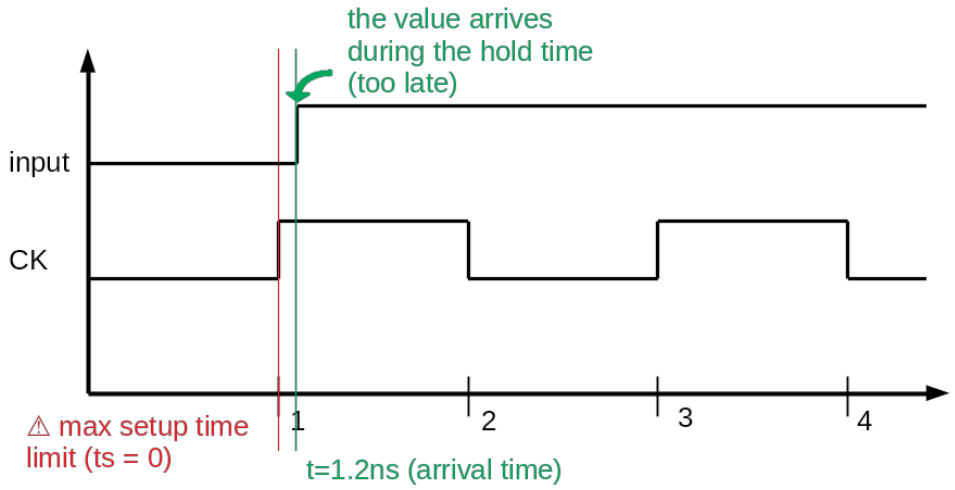
9 Constraints

9.1 Reg-To-Reg

The clock constraint removed the "Generated clocks without clock waveform" issue.

9.2 Input-To-Reg

Let us consider the inputs for the CLOCK domain takes 1.2 nanoseconds to come. Since the clock period in the CLOCK domain is 2 ns (from the addition to our .sdc file in the previous question), it means the arrival time if its signal is 1 ns, for the falling as well as the rising edge. This means that our inputs will always be late to the setup time limit by 0.2 nanoseconds and no inputs will actually be saved in our registers.



Let us actually specify this input delay to our .sdc file and check what the timing report tells us. We can see that we do not have any more "Inputs without clocked external delays".

9.3 Reg-To-Output

We want to constraint the outputs of our design to a delay of 1.2 nanoseconds. After adding this to our sdc file, we can see in our timing report that all our inputs and outputs have delays specified. Unfortunately, when we read the entire timing report we can see, as expected, that there are timing violations.

This time, the worst path starts from the RD_RE register and ends at D_SYNC. The total delay is 12.542 nanoseconds, which is really long. This is due to the definition of our inputs and outputs delays. Our clock definition impose a capture time of 2 ns, which means our critical path, with our specified in/out delays, is too long for our clock frequency. The timing slack is of -10.542 nanoseconds.

10 Report timing