imec
Kapeldreef 75
B-3001 Leuven

Classical Design Flow for tapeout with UMC 90nm technology via Europractice

Labs for ATPG & Power analysis

Author(s):	Geert Vanwijnsberghe		
Date:	7-11-2011	*Doc. No:*	
Keywords:			

System overview (test view)

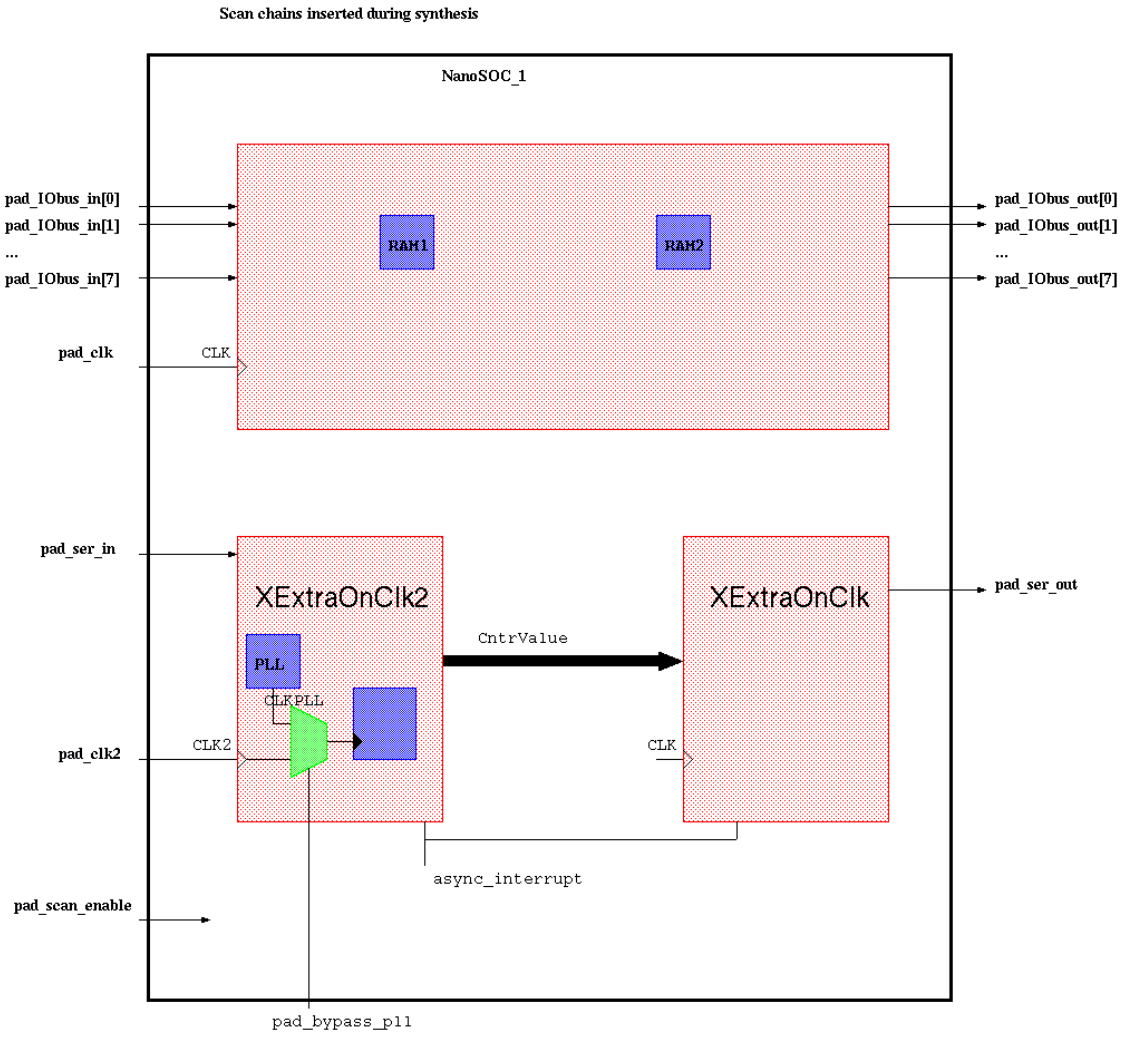


Figure 1 : system overview

Steps done already

- simulation
- synthesis + scan insertion

  9 scan chains were inserted, 8 for the CLK clock domain and one for the CK2 clock domain (cfr. figure 1) . Both share the same scan enable.

Important

- **Look at all Makefiles to see what commands are executed**
- **Have a look at all scripts and ask questions if you do not understand something**
- **The purpose of the lab is to learn about the flow and not just to type make ...**

lab1 : ATPG

In this lab we will first create and verify testpatterns for single stuck at faults. In a second step we will create and verify iddq testpatterns.

The following directory structure will be created during this lab by the 2 tar commands

<your_design>

-hdl <<<< already created during synthesis labs

-libraries <<<< already created during synthesis labs

-synthesis1 <<<< already created during synthesis labs

-synthesis2 <<<< already created during synthesis labs

-synthesis3 <<<< new (will be created with the tar command)

-atpg <<<<<new(will be created with the tar command)

cd <your_path>/<your_design>

cp <source_path>/synthesis3.tgz .

tar xvzf synthesis3.tgz

cp <source_path>/atpg.tgz .

tar xvzf atpg.tgz

have a look at the directory structure

In the directory synthesis3/netlist you find the netlist (corresponding with figure 1) for which we will create the testpatterns.

Part 1 : Generate testpatterns for single stuck-at faults **in the atpg directory**

Have a look at all the files in atpg/scripts. Start with the Makefile. Ask the teacher to help you if you are not familiar with makefiles.

During test the following 4 pins need to be fixed to a constant value:

pad_test =1 : to bypass the rams

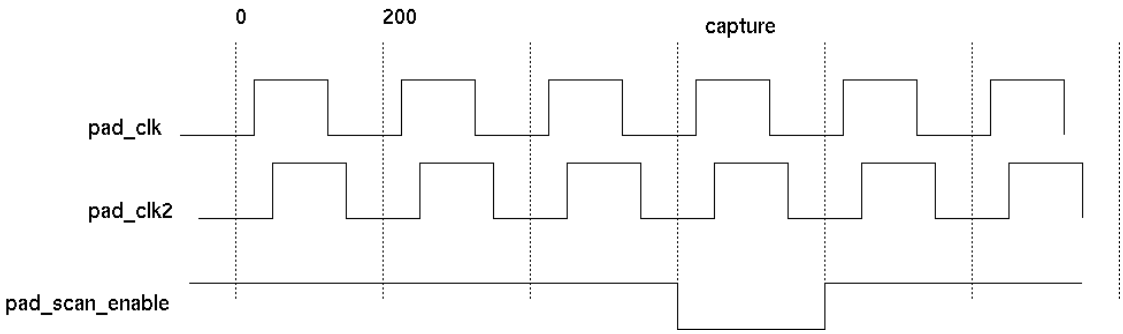
pad_bypass_pll = 1 : to bypass the PLL

pad_pll_PDN=0 : to power down the PLL

pad_pll_TEST=0 : to deactivate PLL test mode

Clock relations during scan

To avoid timing issues during test the following 2 clocks are applied during test .



You should find the above info back in the script files.

Execute now the following commands **in the rundir directory** of the atpg folder :

- **source source.me**
- **make gen_sa**

Have a look to the files in the reports directory. How many patterns are created? What is the fault coverage? Have a look at the undetected fault list.

Part 2: Simulate the patterns for single stuck-at faults

Execute now the following commands **in the rundir directory** of the atpg folder :

- **make sim_sa**

Remark that the default delay (= delay inside the verilog models) is used for all cells in the netlist. This delay does not correspond with the real delay of the cells since the loads are not taken into account. In the next lab on power analysis we will see how the "real or estimated" delays can be annotated to the cells.

Part3: Generate iddq testpatterns

Execute now the following commands **in the rundir directory** of the atpg folder :

- **make gen_iddq**

Have a look to the files in the reports directory. How many patterns are created? What is the fault coverage for this patterns set?

Part4: Simulate/verify the iddq paterns

With the modelsim simulator you could verify that the expected values of the iddq patterns correspond with the simulated ones. This can optinonally be done via : **make sim_iddq_vsim**

If you link however some extra routines to the simulator, you can check if the created iddq patterns are valid ones. This means that for every iddq pattern the simulator will check that eg. no nets are high-impedant or consuming power via pull-up or pull-down resistors. These extra functions ($ssi_iddq) are included in the iddq patterns in verilog format. You may have a look at these in the ../patterns/iddq/verilog/iddq_tb.v file.

The easiest way to verify the iddq patterns is to run a simulation with the vcs simulator of synopsys since the code to be linked with the simulator is already available in the synopsys software tree.

Execute now the following commands **in the rundir directory** of the atpg folder :

- **source source_vcs.me**
- **make sim_iddq_vcs**

Start the simulation and have a look at the iddq_tb.leaky file when the simulation stopped.

Lab 2 : Power analysis

In this lab we will first create meaningfull timing info for all of the cells in our synthesized netlist. In the next step activity files will be created and finally a power analysis will be performed based on the activity files.

The following directory structure will be created during this lab by the 3 tar commands

<your_design>

-hdl <<<< already created during synthesis labs

-libraries <<<< already created during synthesis labs

-synthesis1 <<<< already created during synthesis labs

-synthesis2 <<<< already created during synthesis labs

-synthesis3 <<<< already created during previous labs

-atpg <<<<< already created during previous labs

-gensdf <<<<<new(will be created with the tar command below)

-simulation <<<<<new(will be created with the tar command below)

-powerest <<<<<new(will be created with the tar command below)

cd <your_path>/<your_design>

cp <source_path>/gensdf.tgz .

tar xvzf gensdf.tgz

cp <source_path>/simulation.tgz .

tar xvzf simulation.tgz

cp <source_path>/powerest.tgz .

tar xvzf powerest.tgz

Part 1 : Generate timing info for all cells in the netlist **in the gensdf directory**

In order to be able to simulate a pre_layout netlist (without balanced clock tree) we will generate a timing file for the system (.sdf) in which every flop has some clock to q delay (eg. 0.5 ns) and where all clock gates have zero delay. In this way you can simulate the netlist without getting hold time violations.

Generation of this sdf file is done in the gensdf directory with primetime.

Go to this directory and have a look at the scripts

Execute now the following commands **in the rundir directory** of the gensdf folder :

- **source source_me**
- **make SDF**

Have a look at the generated sdf files

Part 2 : Run a gate level simulation using a generated sdf file in order to produce an activity files (.vcd) (contains changes of all internal nodes)

In this lab we will only run 1 testbench that is loading a specific program “blink” from an

external prom into the program_memory of the Nano_SOC. After this loading the Nano_SOC will execute this program. 2 activity files will be created covering 2 parts of the simulation (reset and reading from prom).

Generation of this activity files is done in the simulation directory with modelsim

Go to this directory and have a look at the scripts.

Execute now the following commands **in the rundir directory** of the simulation folder :

- **source source.me**
- **make gen_gate_activity**

This make take a few minutes.

Have a look at the generated activity files These are a readable ASCII files

Part3 : Perform the power analysis based on the generated vcd files

This is done in the powerest folder

Have a look at the script estpower.tcl

Execute now the following commands **in the rundir directory** of the powerest folder :

- **source source_me**
- **make power**

Important remark:

After layout atpg and power estimations should be redone using the final post-layout netlist and the cell timing based on the .spef files.