

ECE 464 / ECE 520

Tutorial 2: Methodology for Design Analysis

Revision History

January, 2008: Initial version using Modelsim and Encounter (R. Jenkal /P. Franzon)
 January, 2011: Update: 45nm Nangate and Cadence2010 (Z. Yan /P. Franzon)
 January, 2012: Revision from previous version (W. Choi/P. Franzon)
 June, 2014: Update: Procedure (5.2) of power estimation (S. Dey/P. Franzon)
 Sep, 2015: Revision due to tool version update (Junan Zhu, J.Park/P.Franzon)

1. Introduction

The purpose of this tutorial is to provide an understanding of the method for analysis of timing and power for a synthesized netlist. The aim is to provide a realistic power and timing value for the design by running the design through a prototyping flow that would provide a layout based view of the circuit performance. This is done in 3 parts a) automated placement and routing of netlist to determine parasitic (C,R) values b) creation of **switching characteristics (α)** for the nets within the design and c) automated power analysis of the post place and route design using switching activity numbers and realistic **clock frequency (f)** determination. All of these contribute to the final Power (P) = **αCV^2f** .

2. Learning Objectives

- How to setup the directories for the analysis of the netlist.
- How to run the automated place and route flow to create parasitic values for the nets in the design.
- How to create a file that records the switching activity at each net in the placed and routed design.
- How to get the final timing and power values using the annotated parasitic values and switching activity.

THIS TUTORIAL IS BUILD UPON TUORIAL 1.
PLEASE DO NOT CUT AND PASTE EVERY COMMANDS FROM THE
TUTORIAL. THIS LEADS TO ERRORS.
WE ARE EXPECTING YOU CAN INITIALZE TOOLS.

To get a quick overview of overall procedure, you could see appendix A. If you are confused of commands that you have to type, you also look into auto.tcl file.

3. Starting with the analysis flow

The perl code for performing the automated analysis of the designs can be downloaded from http://www.ece.ncsu.edu/asic/share/Tut2/PAD_Flow.pl to a directory of your choice(YOUR WORKING DIRECTORY). All the analysis and design must be performed with respect to this directory (represented by \$WORKING_DIRECTORY).

4. Setting up the directories for analysis

The directory structure that is going to be followed is shown below:

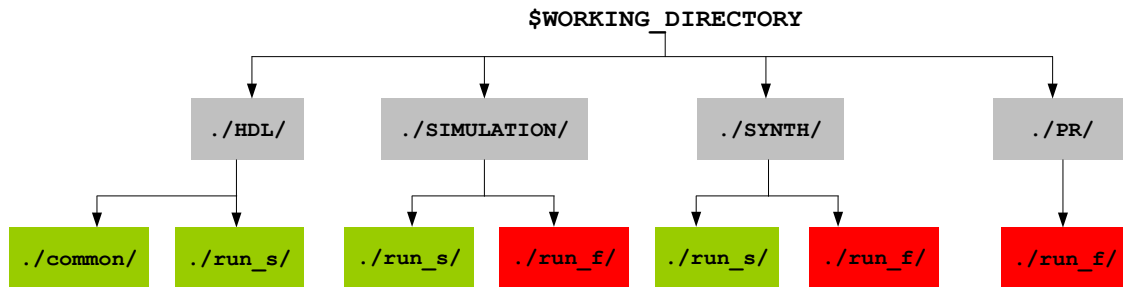


Figure 1: Directory structure to be adopted for design and analysis

In the above, the directories in green are the ones that the student works in while the automated flow works in the directories in red. To set up the above directory structure:

```
> perl ./PAD_Flow.pl -op setup
```

This will setup all of the above directories. At this point please:

- Copy modelsim.ini to ./SIMULATION/run_f and ./SIMULATION/run_s.
- Copy designenv.tcl to ./SYNTH/run_f

The above setup needs to be just once for an entire analysis run. You can do multiple analysis runs for multiple versions of the design using the same directory structure. For the sake of clarity, though, it is advised that you re-create this directory structure for every major change in your HDL. Also note that you are free to run tutorial 1 in the ./SYNTH/run_s, ./HDL/run_s and the ./SIMULATION/run_s directories to create a final synthesizable design.

5. Methodology for analysis of designs

The following tools need to be added to your path before running the flow :
“add modelsim10.3b”, “add synopsys2015”, “add cadence2010”.

- Download <http://www.ece.ncsu.edu/asic/share/Tut2/counter.v> into ./HDL/run_s/ folder.
- Download <http://www.ece.ncsu.edu/asic/share/Tut2/test.v> into the ./SIMULATION/run_f/ and ./SIMULATION/run_s/ folders.

Note that this design is just a larger version of the counter example from Tutorial 1 with a larger register value.

[ASIDE] The ./HDL/common/ folder is best used for storing the header files that define some parameters (for eg. ``define DATA_WIDTH 16`) that will be used to create designs whose characteristics can be modified easily by the manipulation of the values in the above file. This is just a good coding practice and it is left to the student to adopt it if he/she wishes.

At this point, we shall focus on the steps that need to be followed to work with the netlist created by synthesis to get the requisite performance numbers. The entire process is best captured in the figure below (Note that the parts of the flow in the colored boxes are run in an automated manner using the PAD_Flow.pl script):

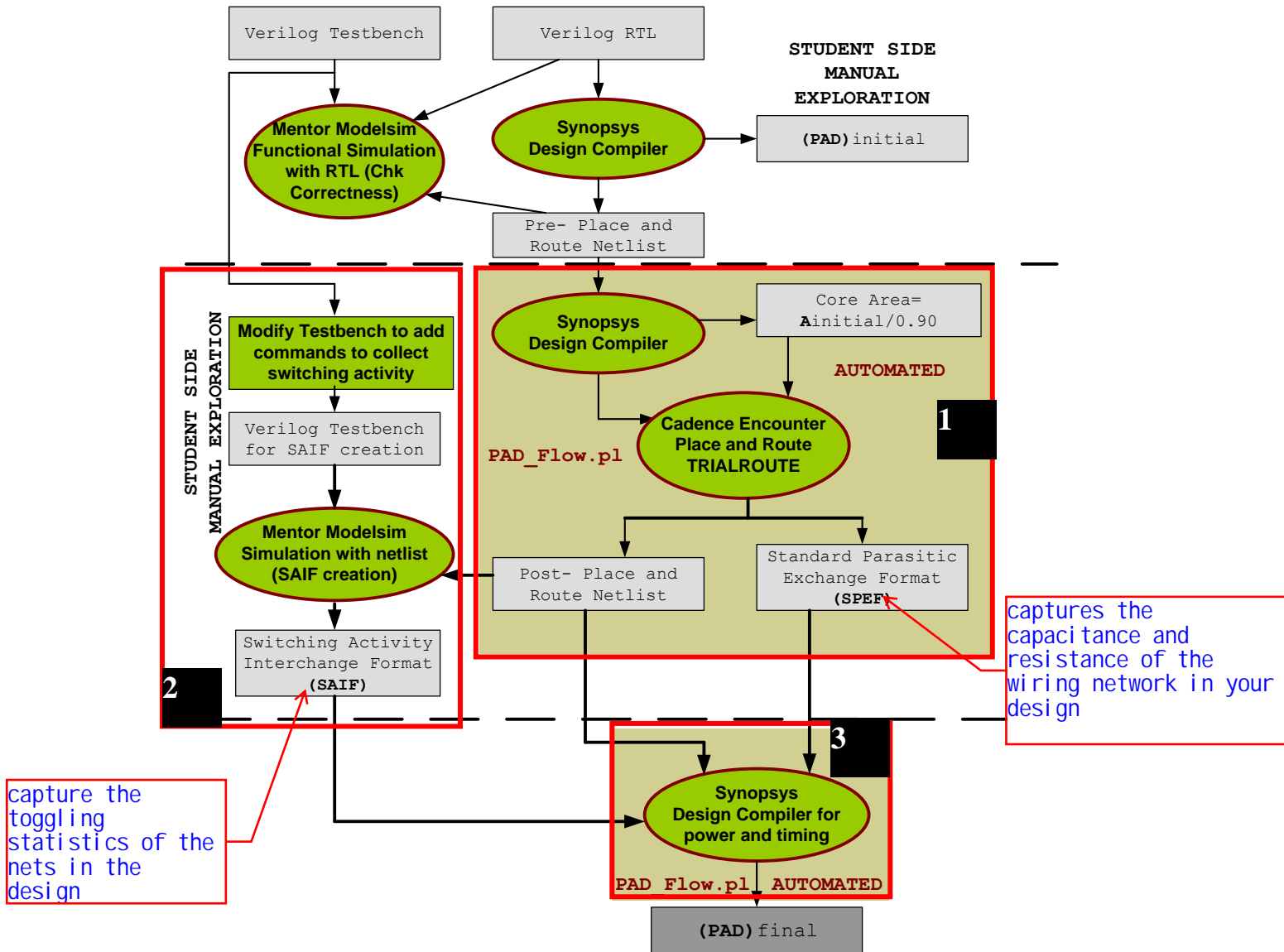


Figure 2: Methodology for analysis of designs

The following steps need to be followed to run the rest of the flow for this example (note you need to do vlib mti_lib if you haven't):

```

> cd ./SIMULATION/run_s
> vlog ../../HDL/run_s/counter.v
> vlog ./test.v
> vsim -novopt test_fixture

```

目前里面包含的文件: modelsim.ini test.v

These commands simulate the counter design. Then you need to synthesize (in `./SYNTH/run_s/`) the final design of choice using the means shown in Tutorial 1. To do this, *all the .tcl files and the .synopsys_dc.setup from Tutorial 1 need to be copied over to ./SYNTH/run_s/ followed by changing the RTL_DIR variable in the setup.tcl script to ../../HDL/run_s to point to the location where all the verilog design files are present.* The *.tcl files are then called in the order shown in Tutorial 1 to create the netlist (`counter_final.v`) within the `./SYNTH/run_s/` directory. After completing the synthesis process, final synthesized netlist (`counter_final.v`) is copied into the `./SYNTH/run_f/` folder.

next time,
from here:
get the
netlist
(counter_fin
al.v)

source synth.tcl

From Figure 2 we see that, starting with the synthesized netlist, there are three major steps that need to be followed to get a final Power (P), Area (A) and Delay (D) value for the design. These three steps are verbalized here for completion and will be performed in the next three subsections (DO NOT RUN THEM AT PRESENT):

1. The `PAD_Flow.pl` script is run to create the Standard Parasitic Exchange Format (SPEF) file that captures the capacitance and resistance of the wiring network in your design. This is explained in Section 5.1.
2. The testbench is copied from `./SIMULATION/run_s/` to `./SIMULATION/run_f/` and modified to be able to capture the toggling statistics of the nets in the design. This information is captured in a Switching Activity Interchange Format (SAIF) file. The method for doing this is explained in Section 5.2. **THIS IS A MANUAL STEP.**
3. The `PAD_Flow.pl` script is run to create final power and delay information for the design using the SAIF and SPEF files. This is explained in Section 5.3.

5.1. Automated run for SPEF creation.

At this point it is assumed that the synthesized netlist (here, `counter_final.v`) for the design has already been created (look at the steps in the beginning of section 5 to determine how this is done) and stored in the `./SYNTH/run_f/` directory. The idea now is to run the `PAD_Flow.pl` script to **work on this netlist** and perform the necessary invocation of tools to create an automated layout for the design under consideration and provide parasitic values for this layout. This requires a) power planning b) placement of the design c) synthesis of the clock tree under skew constraints d) Trialroute of the design to get an estimate for the wire-lengths of the nets in the design and hence their capacitance. This is shown pictorially in slides 5-7 of the accompanying presentation.

Continuing with the example and assuming that the synthesized netlist has the name `counter_final.v`, we can run the analysis flow by doing the following command (in a single line):

```
> perl ./PAD_Flow.pl -op analyze -mod counter -clkname  
clock -period 10 -net ./SYNTH/run_f/counter_final.v
```

The meaning of the different command line options (`-op`, `-mod` etc) can be determined by running:

```
> perl ./PAD_Flow.pl -help
```

For purposes of completeness they are enumerated below:

[NOTE]: Please add modelsim to your path and compile the relevant executables before running the flow

[NOTE]: Ideally you need to do the following: "add modelsim", "add synopsys", "add cadence2010"

[NOTE]: This flow allows for the analysis of the synthesized netlist using post P&R manner

[NOTE]: To run this analysis the inputs needed are:

The Operation Type (-op) [DEFAULT:setup]: setup/analyze/power/memory

The Name of the top level module for synthesis (-mod) [DEFAULT: top]

The full path to the synthesized netlist (-net)

[DEFAULT: ./SYNTH/run_f/top_netlist.v]

The name of the clock port in your design (-clkname) [DEFAULT: clock]

The clock period that is going to be used for analysis (-period) [DEFAULT: 10]

The full path to the saif file (-saif) [DEFAULT: ./SIMULATION/run_f/top.saif]

The Hierarchy to the design instance in your testbench (-inst) [DEFAULT: test_fixture/DUT]

FOR MEMORY BASED ANALYSIS WE HAVE THE FOLLOWING ADDITIONAL INPUTS.
PLEASE IGNORE THESE FOR NORMAL RUN

The Directory where the Memory Generator is run (-memdir) [DEFAULT: ./MEMORY/]

The full path to the memory + top integration (-topfile)

[DEFAULT: ./HDL/run_s/top_with_mem.v]

The name of the instance of the top within the integration file (-topinst) [DEFAULT: top_inst]

The name of the memory generated by the Memory generator (-meminst) [DEFAULT: MemGen_32_12]

ALL THE BEST !!!

[NOTE] When running this step (i.e. -op mod), the only options of significance are "-op", "-mod", "-clkname", "-period" and "-net"

On running the analysis command, information regarding the run and its progress are displayed. You would see the following:

```
##### RUN INFORMATION #####
```

FLOW TO BE RUN WITH THE PARAMETERS:

The Operation Type	(-op)	= analyze
The Module Name	(-mod)	= counter
The Netlist Name	(-net)	=
SYNTH/run_f/counter_final.v		
The Clock Name	(-clkname)	= clock
The Clock Period	(-period)	= 10
The SAIF file	(-	
saif)		= ./SIMULATION/run_f/Top.saif
The Hierarchical Instance of design in testbench	(-inst)	=
test_fixture/DUT		
The Directory where the Memory Generator is run	(-	
memdir)		= ./MEMORY/
The full path to the memory + top integration	(-topfile)	
= ./HDL/run_s/top_with_mem.v		
The name of the instance of the top within the integration file	(-topinst)	=

```

top_inst
The name of the memory generated by the Memory generator (-memname) MemGen_32_12

##### RUN INFORMATION #####

PRESENT WORKING DIRECTORY IS /afs/unity.ncsu.edu/users/z/zyan2/1TA/Tutorial/tut2

#####

      READING NETLIST TO FIND SYNTHESIZED AREA OF DESIGN
      View progress in file "./SYNTH/run_f/dcshell_transcript.out"

#####

TOTAL AREA OF DESIGN TO BE ANALYZED (A): 351.111111111111
RESULTING DIMENSION OF CHIP (smallest multiple of 10 greater than sqrt(A)) = 20

#####

      CREATING DESIGN SPECIFIC CONFIG FILES FOR ENCOUNTER TOOL

#####

      RUNNING ENCOUNTER PLACE AND ROUTE TOOL
      View progress in file "./PR/run_f/encounter_transcript.out"

#####
Done!

```

It is to be noted that the progress of the run can be adjudged from the `./SYNTH/run_f/dcshell_transcript.out` & `./PR/run_f/encounter_transcript.out` files. It is suggested that the latter file be kept open to determine the progress of the encounter tool which might a little time for large designs. It would be best to search for the "ERROR" term in case you do not note progress for a good length of time in the file.

[NOTE] If you do notice errors in the `dcshell_transcript.out` file, it is very likely that you had give the incorrect module name (-mod) and/or path (-net) to the synthesized netlist. You can determine this by looking at the `./SYNTH/run_f/run.tcl` and noting the values assigned to the `read_verilog` and `current_design` commands within it. Remember that the paths in this file are with respect to the `./SYNTH/run_f/` directory in which it is contained. If this step runs correctly, the probability of an error in the Encounter tool is low.

The final files of importance after running this step in the flow can be found in the `./PR/run_f/` directory. And they are:

- `counter_routed.v`: For the general case this file would have the name `<module name>_routed.v` and this corresponds to the netlist of the routed version of the design with buffers put in during clock tree synthesis.
- `counter.spef`: This is the file that contains all the parasitic values for the nets in your design. Again, this file generally takes the name `<module name>.spef`. Note that this file could take a relatively large size.
- There are also timing reports in the above directory which correspond to the delay characteristics of the circuit with and without the annotation of parasitics (`*.rpt`). A

good sanity check would be to ensure that the delay increases as we go from the non-annotated to the annotated case.

We are now ready to get the switching activity information for the nets in the routed netlist.

5.2. Manual run for SAIF creation.

At this point, we should have the netlist for the design (i.e. post place and route) in the `./PR/run_f/` directory. Also, the `modelsim.ini` should already be present in the `./SIMULATION/run_f/` space. Next we need to create a "SAIF File" which contains all the switching activity numbers. To do so, we need to run a simulation using a testbench that encompasses what you consider good benchmark and the netlist in question. This testbench should be copied over to the `./SIMULATION/run_f/` folder. The simulation is going to be performed using Modelsim which you should already be familiar with by virtue of your experience in Tutorial 1.

- To setup the `./SIMULATION/run_f/` directory for simulation type the following commands at the prompt (recollect from Tutorial 1):

The below commands need to be RUN JUST ONCE to set up the work library in a given directory

```
> vlib mti_lib
```

The below 2 commands need to be RUN ONCE FOR a set of simulations (Need not be done if done before in a login session)

```
> add modelsim
```

```
> setenv MODELSIM modelsim.ini
```

- The testbench (`test.v`) now needs to be modified by adding commands that will enable us to collect switching activity statistics. Enable the following codes in your test bench to create a VCD file.

```
$dumpfile ("count.vcd");  
$dumpvars;
```

In this testbench, within the initial block, the following commands are of importance:

- `$dumpfile ("count.vcd");` is used to specify the VCD file name.
- `$dumpvars;` is used to specify which variables should be dumped.

- To compile the design use the following commands (1 line per ">") :

```
> vlog ../../PR/run_f/counter_routed.v
```

```
> vlog
```

```
/afs/eos.ncsu.edu/lockers/research/ece/wdavis/tech/nangate/  
NangateOpenCellLibrary_PDKv1_2_v2008_10/liberty/520/Nangate  
OpenCellLibrary_PDKv1_2_v2008_10_typical_conditional.v
```

```
> vlog ./test.v
```

??

- To invoke the simulator in a command prompt mode and to generate VCD file please use command below:

```
> vsim -c -novopt test_fixture
```

It is to be noted that that "test_fixture" is the name of the testbench module in this case. Replace it with the name of your testbench on creating your own testbenches. The result of this command is the loading of the design of interest and the prompt now becomes **VSIM #>**

- At this prompt (VSIM #>) type the following commands to run the simulation:

```
> run -all;  
> exit; (this might be un-necessary is $finish is reached)
```
- Now use the vcd2saif tool to capture switching activities. Please use command below:

```
> vcd2saif -input count.vcd -instance test_fixture/DUT -output count.saif
```

After successfully execution of the above command, it should generate `count.saif` under `./SIMULATION/run_f`.

We are now ready to run the power analysis flow using the SPEF and the SAIF files.

5.3. Automated run for power estimation

The startup files for this step are

1. The saif file (here `count.saif`) created per **Section 5.2**. It is present in `./SIMULATION/run_f/`.
2. The SPEF file for the design (here `counter.spef`) created per **Section 5.1**. It is present in `./PR/run_f/`.

The idea now is to create an `analysiscommands.tcl` file that would be able to estimate the power consumption of the design within the design compiler environment. To this end, the command that needs to be run is (in `$WORKING_DIRECTORY`):

```
> perl ./PAD_Flow.pl -mod counter -clkname clock -period 10  
-op power -saif SIMULATION/run_f/count.saif -inst  
test_fixture/DUT
```

As seen the major difference between the execution in Section 5.1 is the presence of the `-saif ./SIMULATION/run_f/count.saif -inst test_fixture/DUT` and the `-op power` command line options. The `-saif` option is used to point to the saif file and `-inst` provides the hierarchical name to the design under test in the testbench (here `test_fixture/DUT`).

The above run results in the display of the following message:


```
#####
      CREATING TCL SCRIPT FOR ANALYZING POWER OF DESIGN
#####
```

The important result of running this step is the creation of the power report for the design (counter_pwr_post_annotation.rpt) and the timing report for the design post annotation (counter_timing_post_annotation.rpt). The reports are found in ./SYNTH/run_f. It is also interesting to note the changes in the values within (counter_pwr_base.rpt, counter_pwr_pre_annotation.rpt & counter_pwr_post_annotation.rpt) files.

The base report does not include both SAIF and SPEF, the pre-annotation version includes the SAIF (note the drastic fall in the power values when realistic conditions are used) and the post-annotation version includes both.

<Expected Power Results>

counter_pwr_pre_annotation.rpt:

```
Cell Internal Power = 17.8191 uW (82%)
Net Switching Power = 3.8083 uW (18%)
-----
```

```
Total Dynamic Power = 21.6274 uW (100%)
```

counter_pwr_post_annotation.rpt:

```
Cell Internal Power = 17.8305 uW (72%)
Net Switching Power = 7.0009 uW (28%)
-----
```

```
Total Dynamic Power = 24.8314 uW (100%)
```

It is important to note the increase in the switching power for the nets given that we have increased capacitance and resistance values loaded for the post-annotation report. Also, in this case the increment is not significant given the size of the design. For larger designs, the increase in the net switching power would be much higher.

<Expected Timing Results>

counter_timing_pre_annotation.rpt:

```
data arrival time 9.6297
```

counter_timing_post_annotation.rpt:

```
data arrival time 10.4101
```

We see that the delay does increase on the inclusion of the parasitics from the SPEF file. Another valid sanity check would be to compare the pre-annotated timing report with the timing_max_ttyp_holdfixed_xxx.rpt file from Tutorial 1. The values should match closely given that the critical path should not have changed.

This concludes Tutorial 2. You should have the necessary skills to run a design of your own through this flow and familiarize yourself with those aspects of the invocations that would change from one design to the other.

NOTE: Results may vary based on whether you use “replace_synthetic” versus “replace_synthetic -ungroup” in your synthesis script before calling the flow. It is best to use replace_synthetic -ungroup.

Please go through the accompanying presentation to get a better understanding of the chronology in this tutorial.

APPENDIX A: ADDITIONAL INFORMATION ON FLOW

[INFO: Report Generation Automation]

This tutorial includes two scripts which run all the necessary commands to generate the final reports. These scripts files are “**auto.tcl**” and “**synth.tcl**”. This automation script only intend to help students get the power and timing results fast, to fully understand these scripts, you need to run the all steps stated in this tutorial carefully.

Automation Run Instructions:

1. Download all files from Tutorial 2 on EDA wiki : ASIC Design Tutorials. http://www.eda.ncsu.edu/wiki/Tutorial:ASIC_Design_Tutorials and put all files in one directory. The check-list of files are: “.synopsys_dc.setup”, “auto.tcl”, “CompileAnalyze.tcl”, “Constraints.tcl”, “counter.v”, “designenv.tcl”, “modelsim.ini”, “PAD_Flow.pl”, “read.tcl”, “setup.tcl”, “synth.tcl”, “test.v”. Make sure you get all the files **in one directory.**
2. **Make sure uncomment 2 lines of \$dump... in test.v**
3. Change to this directory, type command “**source auto.tcl**”, all power and timing reports are generated in sub directory ./SYNTH/run_f. The generation process takes about 5 minutes, please be patient, sometimes the screen stops for a while.

[INFO:Section 5.1]:

On running the ./PAD_Flow.pl command, the first step that occurs is the determination of the area of your design by the flow. This is done by reading in the netlist (pointed to using the -net option) into design compiler and creating an area report within ./SYNTH/run_f/ called area_netlist.rpt. The area (in the units of micrometer squared) determined ($A_{initial}$) is used to determine the size of the smallest square with a side which is a multiple of 10 (say, x) with an area $x^2 = A_{final}$ greater than $A_{initial}/0.9$. This forms the dimension of the area within which the automated layout is going to be created. The transcript for this process is generated on screen and in ./SYNTH/run_f/dcshell_transcript.out which can be used to determine correctness of the operation.

With an area estimate, the flow then creates the relevant configuration files for running automated place and route (run.tcl, design.tc, design.conf, clock.ctstch) using design dependent information (module name, clock name, area, netlist path etc). The run.tcl script is then run within the cadence Encounter tool in a command line fashion. This involves the following steps (beyond the scope of this tutorial): floorplan → power planning → placement → power routing → clock tree synthesis → trial route → parasitic information. The final result of this run is the creation of a .spef file that contains all the capacitance and resistance values

of the nets in the design along with the final netlist *_routed.v. The transcript for the entire process can be found at ./PR/run_f/encounter_transcript.out which can be used to check for correctness. The important options in the flow invocation that change with different designs are:

- a) -mod : The name of the top level module that you have synthesized. Let us call it *modname*.
- b) -net : The full path to the synthesized netlist of the new design. It is, if you have followed Tutorial 1, always going to be named *<modname>_final.v*

[INFO: Section 5.2]:

The simulation performed here is performed on the netlist that you is created in the ./PR/run_f/ folder. This is done to capture the number of times all the wires (nets) in the netlist switch from 1→0 and 0 → 1 (this is called switching activity number). This allows the independent determination of the fall and rise power for each net. To enable simulation, the netlist and the standard cell library verilog need to be compiled (using vlog in this case). This is because the netlist represents the design in terms of the cells in the standard cell library and both verilog representations need to be compiled. In addition to this, the testbench needs to be modified to be able to capture the switching activity numbers of all the nets using Modelsim. The \$toggle_x() commands enable this to be done along with the invocation of the simulator with a -pli option. Of particular importance is the need for the path to the libvpower.so to be changed with the type of operating system under use. The important options that change with different designs are:

- a) Change call to \$set_toggle_region(**<test fixture name>.<top level instance name>**): This reflects a new, if applicable, a new testbench name (presently test_fixture) and **instance** of the design under test (presently DUT).
- b) Change call to \$toggle_report("**<modname>.saif",1.0e-9,"<test fixture name>.<top level instance name>**"): This reflects a new **top level module** name (presently counter). Additionally, a new testbench name and instance of the device under test need to be used in the same manner as step a).
- c) Change vlog ../../PR/run_f/**<modname>_routed.v** : This, again, reflects a new **top level module** name (presently counter) for compilation.
- d) Change call to vsim -novopt -c -pli ... **<test fixture name>** to reflect a new testbench name.

[INFO:Section 5.3]:

With the saif and spf file having being created, the next step is to create the final timing and power values for the netlist under consideration. To achieve this, a ./SYNTH/run_f/analysiscommands.tcl file is created which points to the correct input files and the necessary constraints to create the power and timing values. This file is then sourced within design compiler by the Flow. To enable this flow to be run on a different design, it is important to make the following changes:

- a) -op: This changes to "Power" from "Analyze"
- b) -mod: already explained
- c) -saif: Full path to the saif file. This needs to change to reflect, the new **<modname>.saif** file.

- d) `-inst`: This reflects the hierarchy to the top level instance name `<test fixture name>/<top level instance name>` (remember the “/”). This is similar to step a) in the previous section.

The final timing and power results of importance are going to be:

`<modname>_clock_pwr_post_annotation.rpt`


`<modname>_pwr_post_annotation.rpt`

`<modname>_timing_post_annotation.rpt`

APPENDIX B : VIEWING FINAL LAYOUT

For those interested, the final design would look something like the figure shown below. To be able to invoke this you would need to do the following

- Ensure that the encounter tool is accessible (i.e. an “add cadence2010” was performed previously)
- Go to the `./PR/run_f/` folder
- Type “encounter”. This opens up a GUI and a prompt of the form `encounter #>`
- On the prompt type the following
 - `encounter #> loadConfig design.conf`
 - `encounter #> restoreDesign . counter_cts_trialroute.enc.dat/counter`

You can now view the resulting design with wiring by hitting the  button (fit). See figure 3 for the final layout.

APPENDIX C: FREQUENT PROBLEMS AND THEIR SOLUTIONS

Problem #1:

I get the error: Can't read `link_library` file 'your_library.db'

Solution:

It is likely that the `.synopsys_dc.setup` file has not been copied over into the `SYNTH/run_s/` folder. Also, it is possible that the filename is missing the “.” at the beginning of the file.

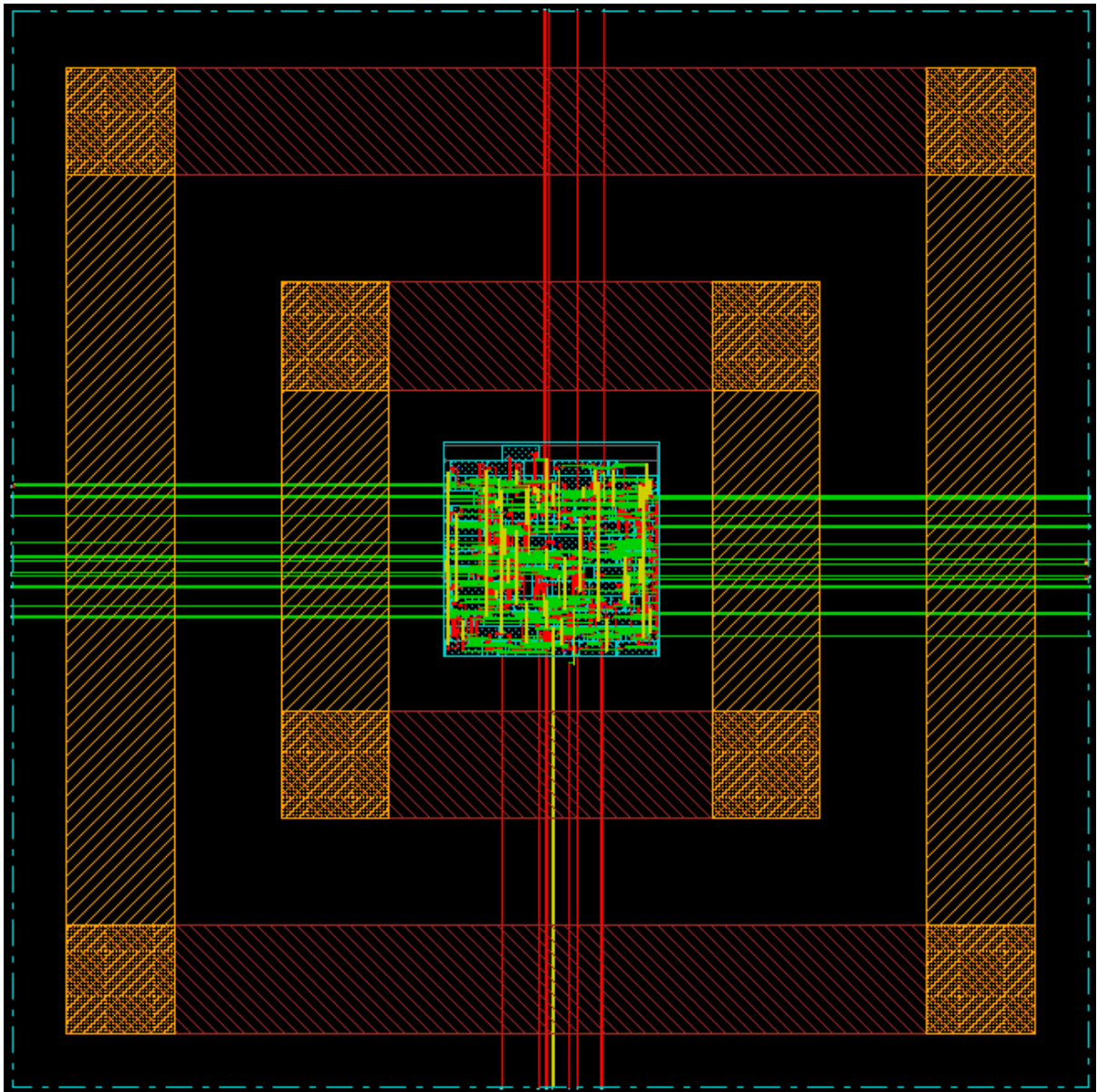


Figure 3: Final Layout Created.

Problem #2:

I cannot find any output files. Where should I be looking?

Solution:

The timing and power reports should be in the `./SYNTH/run_f/` folder.

Problem #3:

The `vsim -novopt -c -pli $SYNOPSIS/linux/power/vpower/libvpower.so test_fixture` command gives me the following error:
`# ** Error: (vsim-3043) ./test_switching.v(xx): Unresolved reference to 'DUT'.`

Solution:

Please check the name of the instance of your design in the test fixture. The tool is not able to find the instance DUT in your design. Replace the name within the `$toggle_x()` commands with the correct instance name of your top level design.

Problem #4:

When I go through the tutorial I get this error:

```
#####
RUNNING ENCOUNTER PLACE AND ROUTE TOOL
View progress in file "./PR/run_f/encounter_transcript.out"
#####
sh: encounter: command not found
```

Solution:

You need to do an "add cadence2010" before running the Flow.

Problem #5:

Since `counter.v(HDL/run_s)` and `test.v(SIMULATION/run_s)` are in different directories, how can i create a common vlib?

Solution:

Just Point to them from the directory you are in. There is no need for all your files to be in one directory. Assuming you want to do a simulation of your verilog files before running it through the flow: work in `./SIMULATION/run_s` and compile as

```
vlog ../../HDL/run_s/counter.v
vlog ./Test.v
```

Problem #6:

Are these results valid? I got negative values for the data arrival times.

```
...
data arrival time -1.65
-----
slack (MET) x.xx
```

Solution:

There should be another data arrival time result in the report before the one you are looking at. In this case (the second data arrival time result), the arrival time is being subtracted from the required time to get the slack.

Problem #7:

I am getting the following error when I attempt to run the SAIF creation step:

```
vsim -c -pli /ncsu/synopsys/linux/power/vpower/libvpower.so ?novopt
test_fixture
** Error: Failed to find design unit work.?novopt.
# Error loading design
```

Solution:

There is a copy paste problem here which is making the tool look for novopt as a design unit. Make sure that you are typing in "-novopt" (No copy and paste). If this option is present, the "*** Note: (vsim-3812) Design is being optimized..." message would not be printed out. Please do not copy and paste.