EECS 151/251A ASIC Lab 6:
Power and Timing Verification

Written by Nathan Narevsky (2014,2017) and Brian Zimmer (2014)
Modified by John Wright (2015, 2016), Ali Moin (2017) and Arya Reais-Parsi (2019)

Overview

So far, you have learned how to translate a design described in Verilog to a fully-routed design. Before this design can be manufactured ('taped-out') there are two last steps that you will learn in this lab—power analysis and timing verification. Power analysis will determine how much energy your design will consume, and timing verification will ensure that no hold time violations exist in the design.

To begin this lab, get the project files by typing the following command

```
git clone /home/ff/eecs151/labs/lab6
cd lab6
```

If you have not done so already you should add the following line to your bashrc file (in your home folder) so that every time you open a new terminal you have the paths for the tools setup properly.

```
source /home/ff/eecs151/tutorials/eecs151.bashrc
```

Note: after doing this, you need to manually source your own .bashrc file or open a new terminal. You can test whether your path is setup properly or not by typing `which dc_shell` and making sure that it does not say `no dc_shell in ....`

Setting Up the Other Tools Required for This Lab

In this lab, we will be using a tool called PrimeTime in order to look at timing and power analysis of our design. In order to run this tool, we have to run through the previous tools (synthesis and place and route) so let's go ahead and do that now.

The commands below will run through the synthesis flow:

```
cd dc-syn
make
```

Next we need to place and route the design. This can be done with the following commands:

```
cd ../icc-par
make
```

Generating a VCD and SAIF File

In order to get more accurate information for power analysis, we can use information from a simulation of our design within the tool. You can do so by going into the `vcs-sim-gl-par` folder. If you look at the Makefile in this folder you will see a new section compared to previous versions replicated below:

```
convert_saif = vcdplus.saif

$(convert_saif): %.saif: %.vpd
    vpd2vcd $(patsubst %.saif,%.vpd,$@) $(patsubst %.saif, %.vcd, $@)
    vcd2saif -input $(patsubst %.saif, %.vcd, $@) -output $@
    date > timestamp

convert: $(convert_saif)
```

This section will take the `vpd` file, convert it to a `vcd` file and also convert the `vcd` file to a `saif` file. A `vcd` simply contains the time information about the signals, in terms of when they transition and what their new and old values are. The `saif` file contains switching activity of the individual nets, as well as the amount of time that the nets are either 1 or 0. An example signal from the `saif` file looks like below (taken from line 1436):

```
(operands_bits_A\[10\]
  (T0 27639811) (T1 2080189) (TX 0)
  (TC 54) (IG 0)
)
```

This is entry in the file for the `operands_bits_A[10]` register. The `T0` is the duration of time in the logic 0 state, `T1` is the time in logic 1, `TX` is the time in an unkown state, `TC` is the total number of transitions (rising and falling combined), and `IG` is the number of transition glitches during the monitoring.

While you are in the `vcs-sim-gl-par` folder type the following:

```
make convert
```

This will run through a post-place-and-route simulation and create the necessary files for more detailed power analysis.

Running Primetime

After generating the switching activity, Primetime is run from the following folder:

```
cd ../pt-pwr
```

Now we are going to run through PrimeTime to analyze the power and timing of our design. As we have learned in previous labs, a very useful way to debug the scripts is to place the `return` command inside a script, which allows you to interactively debug during one of the steps.

Open `pt_scripts/pt.avg.tcl`, and on line 18, add `return`. Then run:

```
make
```

This will create the necessary files and directories to run through Primetime. It will also load and setup necessary files and variables for running through power and timing analysis.

This should return you to an interactive Primetime shell where we can start inputting commands. We are going to start with these few lines below:

```
set power_enable_analysis true
set power_analysis_mode averaged

set report_default_significant_digits 4 ;
set sh_source_uses_search_path true ;
set search_path ". $search_path" ;
```

Going through these lines, the first one enables power analysis. In order to run any sort of power reporting (which we will be doing later) you need to set this flag. The `power_analysis_mode` sets how the tool does the calculation for power. Averaged, as set here, is the default, and will only calculate the power based on toggle-rate and state-probability. There is also a `time_based` option that will calculate power as a function of time based on waveforms from a simulation that needs to be setup properly. We will look into this more later in the lab.

Setting the significant digits simply truncates numbers for cleaner printing. The next line allows files to be found in the search path that is created in some of the setup scripts, and the last line here adds the current folder to the search path. Continuing, we need to issue the following commands:

```
set link_path "* $link_path"

read_verilog top.output.v
current_design top
link
```

This will setup the link path, which is different from the search path in that it is only used for the `link` command, which has the same purpose as it did in Design Compiler. The `read_verilog` command will read in verilog files, which in this case is the output from IC Compiler.

```
read_parasitics -format sbpf ../../icc-par/current-icc/results/top.output.sbpf.max
```

This command will read in the `Synopsys Binary Parasitics` file which is generated from IC Compiler to denote the different parasitic resistances and capacitances in the design. Keeping it

as a binary file greatly reduces the file size for large designs, but unfortunately means that it is no longer human-readable. The fact that the filename has the word `max` in it indicates that it is the worst case parasitics, which is what we would be concerned about for the critical path.

```
read_sdc -echo ../../icc-par/current-icc/results/top.output.sdc
```

Now we read in the `Synopsys Design Constraints`, which are very similar to the constraints that were originally given to the tools when we ran them. Take a brief look at your `sdc` file and it should look something like below:

```
set sdc_version 1.9

set_units -time ns -resistance MOhm -capacitance fF -voltage V -current uA
set_propagated_clock [get_pins gcd/pll/CLK_4X]
set_propagated_clock [get_pins gcd_io/PAD_pll_ref_clk/DOUT]
create_clock [get_pins gcd/pll/CLK_4X] -name fast_clk -period 0.69 -waveform {0 0.345}
set_clock_uncertainty 0.0345  [get_clocks fast_clk]
create_clock [get_pins gcd_io/PAD_pll_ref_clk/DOUT]  \
   -name slow_clk  -period 20  -waveform {0 10}
set_clock_latency 0.15  -clock [get_clocks slow_clk] \
   [get_pins gcd/GCDdpath0/clk_gate_B_reg_reg/latch/GCLK]
set_clock_latency 0.15  -clock [get_clocks slow_clk] \
   [get_pins gcd/GCDdpath0/clk_gate_A_reg_reg/latch/GCLK]
set_clock_groups -asynchronous  -name fast_clk_1  -group \
   [list [get_clocks fast_clk]]  -group [list [get_clocks slow_clk]]
set_timing_derate -late -net_delay  1.01
set_timing_derate -early -net_delay  0.99
set_timing_derate -late -cell_delay 1.01 [current_design]
set_timing_derate -early -cell_delay 0.99 [current_design]
```

If you look at the constraints that were passed into IC Compiler in `../icc-par/setup/constraints.tcl` you should be able to tell where most of these constraints are derived from. Others come from different files, but the main point here is that it sets up all of the clocks and units for everything. After we have read in the parasitics and constraints, we can move on to update and check the timing:

```
update_timing -full

check_timing -verbose
```

If you get a warning saying that there are endpoints that are not constrainted for maximum delay, do not worry - these paths have to do with the crossover between clock domains and we told the tools to specifically ignore these paths.

To report the timing of the paths use the following command:

```
report_timing -delay min_max -input -net -sign 4 -nosplit
```

Please note that this will return both the minimum (hold time) and maximum (setup time) paths.

Timing Analysis in Primetime

Question 1: Timing analysis

a) Compare the critical path (in the fast_clk domain) as reported by Primetime to the same exact path in ICC. How different are the results?

b) Variation in transistor delay inside gates can cause hold time even if the design passes timing. How much hold time slack is in your design?

Question 2: Timing analysis

a) Explore VCD file by opening `vcdplus.vcd` in DVE. Hypothetically, can the information in these waveforms be used to measure energy due to glitching? Hint: Look into the signals contained within the `gcd/GCDdpath0` block. Submit a screenshot explaining why or why not.

b) Explore SAIF file by opening `vcdplus.saif`. What is the switching activity of the `clk` net? What is the switching activity of `result_bits_data[2]`? Using the `DURATION` and `TIMESCALE` information, what is the clock frequency of the simulation (and how did you derive it)?

Power Analysis in Primetime

Primetime can estimate the power your design will consume. There are two ways to analyze power in Primetime—time-based and averaged. The steps you went through in the first part of the lab explored average power analysis. Remove the `return` you added earlier to `pt_scripts/pt.avg.tcl`, and run the following:

```
cd pt-pwr
make pt-avg
```

This will run the averaged power analysis, which is basically the previous set of commands with the following syntax for a `report_power` command:

```
report_power -nosplit -verbose -hierarchy
```

To run time-based analysis, interrupt the script before it exits and open the GUI.

```
cd pt-pwr
vim pt_scripts/pt.time.tcl  (and replace 'exit' on the last line with 'return')
make pt-time
gui_start
```

The difference between these files is that the averaged script reads in a `saif` file, which we learned about earlier in the lab. The time-based power script reads in a `vcd` file, which is the time domain representation of the waveforms from a simulation. This allows the tool to calculate power as a function of time and should be able to plot it, but unfortunately our liscense for that is not working.

Power is split into two categories: leakage power and dynamic power. Primetime further divides dynamic power into two categories: internal power and switching power. Internal power contains all power dissipated within the cell boundary (this includes both short-circuit current and charging intrinsic capacitances). Switching power includes the power dissipated by driving wire capacitance and the input capacitances of the gates. From inside the gui, let's take a look at the breakdown of our design by different types of power. To do this, go to Power and click on "Show Power Analysis Driver" which should pop up a window like below:

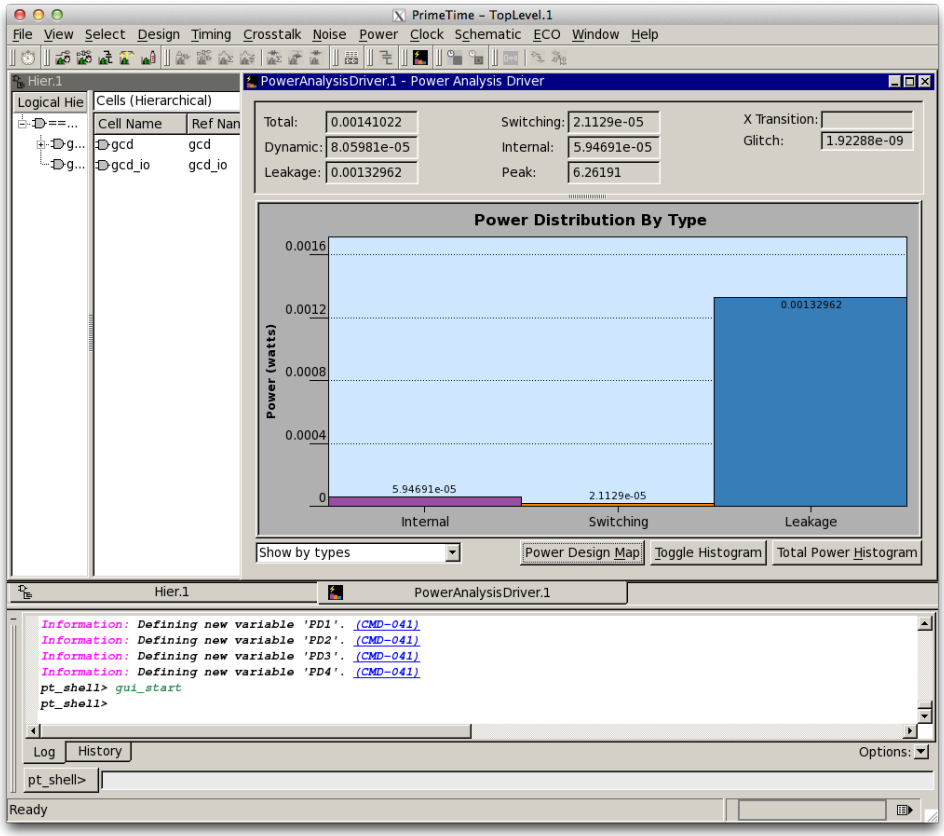

Figure 1: Power Analysis Driver

Since this is a very small design that we are running very slowly in the testbench, the majority of the power is leakage, as seen here. For each of these types of power we can see how the power is broken down by sub-blocks, so let's look at an example of that now. Go to Power again and click on "New Power Design Map" and select "Total Power Density". You should see something similar to below after increasing the View Level to 4.

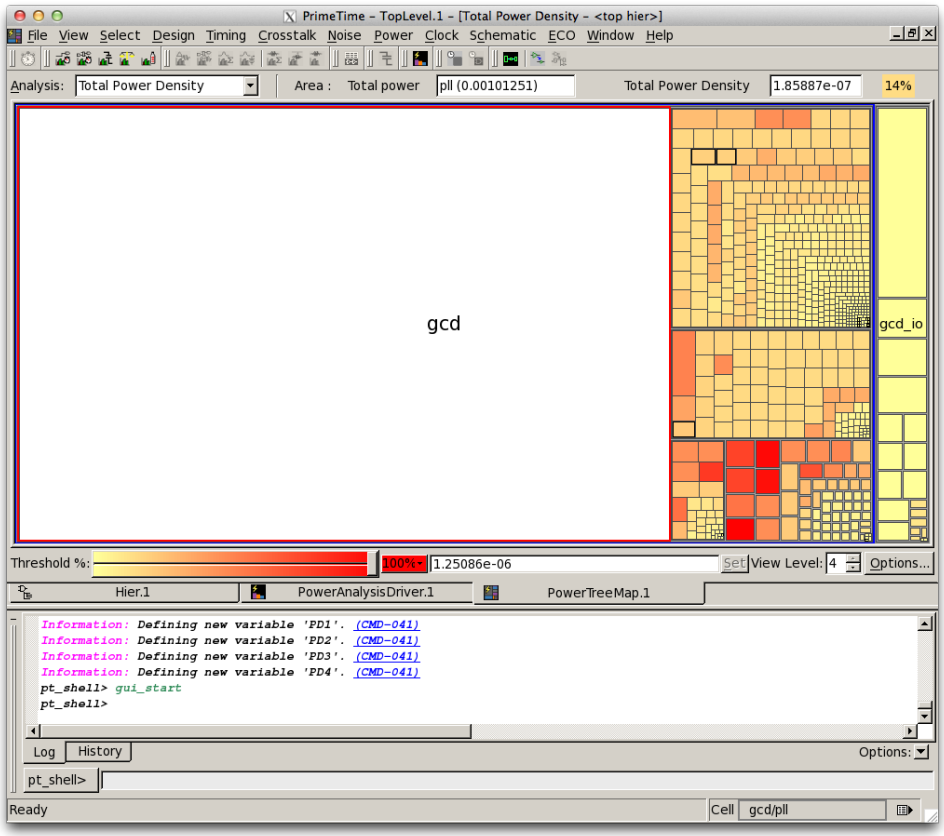

Figure 2: Power Tree Map

There are options to select by different types of power, and it shows you the sub-blocks to multiple levels of hierarchy. The coloring should be based on the total power, and you can change what that threshold is for visual purposes. Double-clicking on a block will zoom into that hierarchy, and you can return one level by right-clicking and selecting `Pop`.

Question 3: Power analysis

Power analysis of the final place-and-routed design will closely match reality, but requires going through every step in the flow. It is possible to measure power before placement even begins by measuring the power of the design after synthesis. In this problem, you will learn why the power results differ between post-synthesis and post-routing measurement.

a) Make a table comparing the post-synthesis power report to the post-routing power report. To do this, follow this suggested procedure: Generate a VCD and SAIF activity file from your post-synthesis design by modifying the Makefile in the `vcs-sim-gl-syn`. Copy the `pt-pwr` directory to `pt-pwr-syn`. Modify the Makefile so that Primetime uses the post-synthesis design instead of the post-place-and-route results (modify the `icc_v` and `icc_sdc` variables point to the appropriate results form `dc-syn`). Also modify the Makefile to use the switching activity from this design (modify the `vcs_dir` directory). Compare leakage, switching, and internal power.

b) Modify the testbench to replace the current GCD test vectors with different vectors (eg. compute the GCD of different A/B than before). Re-run power analysis, and report how much the leakage and dynamic power changed.