Place & Route Tutorial #2

In this tutorial you will use Cadence Encounter and Synopsys Prime Time to synthesize a clock-tree and verify the timing for a simple ARM microprocessor. This tutorial assumes that you have worked through *Place & Route Tutorial #1*, which introduces Cadence Encounter and that you know how to perform the base-line physical design flow on flat designs. Note that the timing values that you will see when you run through this tutorial will very likely be different from the values in this document, and that's to be expected. It's very hard to get exactly the same design twice, because small differences in the way you run the tools can have a large effect on performance. Take the values in this document as a guide for how to interpret the results of the tools, and try to make similar interpretations with your own values.

I. Setup

- Log in to a Linux or Solaris machine.
- Download and unpack the file **pr_tut2.tar.gz**. This archive contains a directory called "pr_tut2" subdirectories called "v" and "pr" with files needed to complete this tutorial.
- Copy the v/sim and v/src/rtl direcotries from the **cortexmods.tar.gz** distribution into this directory tree.

II. Synthesize the Design

Change to the v/synth directory and look at the **setup.tcl** file. You'll notice that this file has been modified for you to include the top-level module name and clock port. A clock period of 40 ns has also been annotated. Likewise, the **read.tcl** file has been updated to include the filenames for the ARM core.

Next, look at the Constraints.tcl file. You'll notice the following two lines:

```
set CLK_SKEW 0.05
set_clock_uncertainty $CLK_SKEW $clkname
```

These lines specify the amount of clock skew (50 ps) that will be assumed during synthesis. This clock uncertainty can lead to hold-time violations. If hold time violations occur, then one way to fix the problem is to increase the delay to short paths by adding inverters or buffers. These "hold-fix buffers are added in the CompileAnalyze.tcl script with the following two lines:

```
set_fix_hold $clkname
compile -only_design_rule -incremental
```

We will check the quality of the final routed design by its ability to meet the overall hold-time and setup-time constraints in *Synopsys PrimeTime*. Go ahead and synthesize the design using the commands given in the README.txt file. When finished, examine the file **timing_min_fast_holdcheck_tut2.rpt**. You should see that the hold-time constraints have been met with a slack of less than 1 ps.

III. Place & Route the Design

Change to the ../../pr directory and look at the files design.conf and design.tc. You'll notice that these files have been modified for the CORTEXM0 design. Go ahead and run the flow through to the *cts* step.

Open up the **clock.tmpl** file and take a look. This is the primary command file for clock-tree synthesis. You should find the following lines:

```
AutoCTSRootPin HCLK
Period         40ns
MaxDelay       40ns # default value
MinDelay       0ns   # default value
MaxSkew        300ps # default value
SinkMaxTran    400ps # default value
BufMaxTran     400ps # default value
```

These lines show that the root pin, period, and max insertion delay have been taken from the design.tc file. The skew and transition times remain unspecified, however. It's somewhat inconsistent for the skew to be 300ps here and 50ps for *DesignCompiler*. We might get a lot more skew than expected and suffer hold-time violations. But let's leave it as it is for now and see what happens. Copy **clock.tmpl** to **clock.ctstch** and re-run the *cts* step.

After clock-tree synthesis completes, you should be able to open up the **clock.ctsrpt** file to see a report of the output. Open this file and you'll see the following:

```
Nr. of Subtrees                : 1
Nr. of Sinks                   : 841
Nr. of Buffer                  : 8
Nr. of Level (including gates) : 2
Root Rise Input Tran           : 120(ps)
Root Fall Input Tran           : 120(ps)
Max trig. edge delay at sink(R): u_logic_Txj2z4_reg/CK 453(ps)
Min trig. edge delay at sink(R): u_logic_Rkd3z4_reg/CK 343.6(ps)
```

These lines show that 8 total buffers were added in the clock tree to drive 841 total clock sinks. It also shows that the skew predicted prior to routing is about 110 ps. That's well within our design goals. Further down in the file, you'll see the following lines:

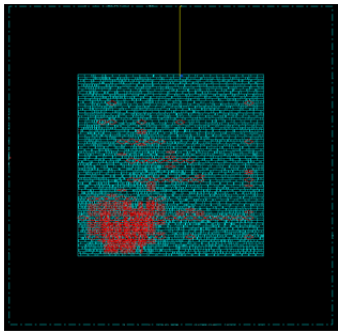
```
                                (Actual)                (Required)
Max. Rise Buffer Tran          : 158.1(ps)              400(ps)
Max. Fall Buffer Tran          : 55(ps)                 400(ps)
Max. Rise Sink Tran            : 265.6(ps)              400(ps)
Max. Fall Sink Tran            : 80.9(ps)               400(ps)
Min. Rise Buffer Tran          : 158.1(ps)              0(ps)
Min. Fall Buffer Tran          : 54.9(ps)               0(ps)
Min. Rise Sink Tran            : 157.2(ps)              0(ps)
Min. Fall Sink Tran            : 63.7(ps)               0(ps)
```

These lines show that the rise times at the clock-sink inputs are within a factor of 2, compared to the rise times at the buffer inputs. Still further down in the file, you'll see the following:

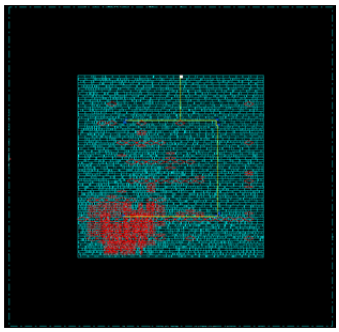
```
  Main Tree from HCLK w/o tracing through gates:
     nrSink : 841
     nrGate : 0
     Rise Delay [343.6(ps)  453(ps)] Skew [109.4(ps)]
     Fall Delay [392.7(ps)  444.3(ps)] Skew=[51.6(ps)]


HCLK (0 0) load=0.0198927(pf)

HCLK__L1_I0/A (0.0012 0.0012)
HCLK__L1_I0/ZN (0.2447 0.1157) load=0.122234(pf)

HCLK__L2_I6/A (0.2618 0.1328)
HCLK__L2_I6/ZN (0.3412 0.3948) load=0.125775(pf)

HCLK__L2_I5/A (0.2575 0.1285)
HCLK__L2_I5/ZN (0.4329 0.4259) load=0.203255(pf)

HCLK__L2_I4/A (0.2623 0.1333)
HCLK__L2_I4/ZN (0.3452 0.3966) load=0.128539(pf)

HCLK__L2_I3/A (0.2565 0.1274)
HCLK__L2_I3/ZN (0.4443 0.4285) load=0.21335(pf)

HCLK__L2_I2/A (0.2533 0.1243)
HCLK__L2_I2/ZN (0.35 0.3927) load=0.13969(pf)

HCLK__L2_I1/A (0.2657 0.1367)
HCLK__L2_I1/ZN (0.3402 0.3969) load=0.121787(pf)

HCLK__L2_I0/A (0.2535 0.1245)
HCLK__L2_I0/ZN (0.3602 0.3966) load=0.14782(pf)
```

These lines finally hint at the structure of the tree. We see that the first level of the clock tree has one buffer, and the second level has seven. Presumably, each of these last-stage buffers drives about 120 clock sinks. To verify this fact, you can look at the verilog file output by either the *trialroute* or *route* step. Having so many sinks per buffer is not necessarily the best way to build a clock-tree. Because delays vary so strongly with transition time, and because standard-cell libraries are typically characterized for small loads and transition-times, it's generally better to keep these times small. In this case, however, the predicted times are well within the constraints that were set, so the CTS tool is minimizing the complexity of the tree needed to give us the characteristics we wanted. If we want tighter control over the transition times, we'll need to set tighter constraints in the **clock.ctstch** file.

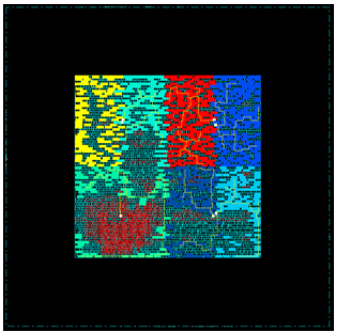
Pouring over the **clock.ctsrpt** file can be cumbersome. If you want to get a quick, graphical look at the clock tree, you can open the **CORTEXM0_cts.enc** design in *Encounter*. Switch to the Physical View and choose **Clock → Display → Display Clock Tree**. Then, in the "Display" section of the pop-up window, select "Display Clock Tree" and "Selected Level," set the level to 1, 2, or 3, and click "Apply". Turn off the trial-routed "Nets" and "Special Nets" by de-selecting them in the Layer Control panel on the right. Use Ctrl-R to redraw the screen (note that the red congestion-markers still remain). You should see each level of the tree displayed as shown below. The last level highlights the sinks, giving a common color to sinks that share the same last-stage buffer.



Level 1

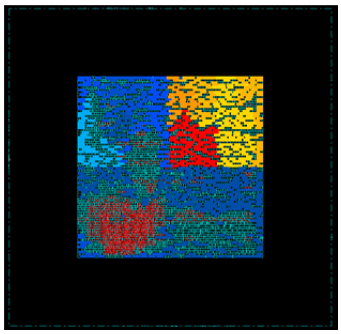


Level 2

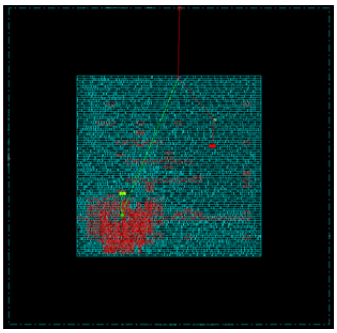


Level 3

You can graphically display the skew in the clock-tree by choosing "Display Clock Phase Delay" in the **Display Clock Tree** dialog, as shown below. Sinks with the longest collective insertion delay are colored red, while sinks with the shortest collective insertion delay are colored blue. To see the sinks with the longest and shortest individual insertion delays, along with their paths highlighted, choose "Display Min/Max Paths" in the **Display Clock Tree** dialog, as shown below. .



Phase Delay



Max, Min Paths

Note that a global-route/trial-route was executed as part of the optDesign command in the run_cts step. You should see the following line towards the end of the **run_cts.log** file. This line gives us a rough measure of how congested the design is.

```
Overflow: 2413 = 598 (3.39% H) + 1815 (10.29% V)
```

Run the *trialroute* step before proceeding. Is it somewhat redundant, because a trial-route has already been performed, but the *trialroute* step will also generate a Verilog netlist and extract the parasitic resistances and capacitances that we need for the next step.

IV. Analyze the Timing in *PrimeTime*

Change back to the ../v/synth directory and examine the **run_pt.tcl** file. Note that the clock node, module name, and clock period have been set for you. As with the ECE 520 Tutorial #1, this script also sets the drive-strength of all inputs (except the clock) to be the Q pin on a size 1 D flip-flop. The input delay is set to the clock-to-Q delay of the flip-flop (for the slow timing corner). What's different about these input settings is that the insertion delay of the clock is also added to the input delay. This is because *PrimeTime* will calculate the insertion delay for each sink when checking timing constraints. If the insertion delay is not added to the input delay, then inputs can arrive well ahead of the clock (which can be a rather serious hold-time violation). Since we're not so concerned with the timing of inputs in this tutorial, you can leave this value as it is for now (1 ns). Note also that the output delays of all outputs are set to zero, because the delays to these outputs would not be calculated at all, unless some timing constraint were applied to them.

Next, note the two variables type and corner. The **type** variable can be set to either *routed* or *trialrouted*, depending on whether you want to get parasitic capacitances from the *route* or *trialroute* step, respectively. The corner variable should be set to either *fast* or *slow*, depending on which timing library you want to use. Right now, these variables are set to *routed* and *fast*. Set the **type** to *trialrouted* before proceeding.

See the **README.txt** file for the list of commands needed to run the timing check. Note that *PrimeTime* reads the setup file **.synopsys_pt.setup** to define its paths and other variables. Luckily, this file is not very different from the *DesignCompiler* setup, so we will simply copy the **.synopsys_dc.setup** file before running *PrimeTime* for the first time. Note also that you can run a signal-integrity check by simply using the **run_ptsi.tcl** file in place of the **run_pt.tcl** file. For simplicity, we'll skip running signal-integrity analysis and remain with the **run_pt.tcl** file for the remainder of this tutorial.

Run *PrimeTime* and examine the file **timing_pt_fast.rpt**. This file contains hold-time and setup-time checks. You'll notice that the setup-time check has been met with a slack of 36.5 ns. Next, look at the file **timing_pt_fast_trialrouted.rpt** and you'll see that the setup-time slack is a bit smaller: 35.6 ns. The first file is a timing report before back-annotation of parasitics (the SPEF file). It's good practice to compare these files at least once, just to ensure that the parasitics were read correctly. This is one of the most common places that errors occur. In this case, I would expect the difference between pre-route and post-route slack to be much larger, which may be an error in the timing analysis, or may simply indicate the pre-route wire-load models are working well. Let's assume that it's correct for now.

Now set the corner to *slow* in the **run_pt.tcl** file and re-run *PrimeTime*. Check the setup-time slack in the **timing_pt_slow_trialrouted.rpt** file, and you'll see that the setup-time slack is 12.63 ns, which is positive, so things are looking good so far.

Now let's look at the clock properties. Examine the **timing_pt_slow_clock.rpt** file, and you'll see that *PrimeTime* predicts max and min insertion delay (called "maximum setup launch latency" and "minimum setup capture latency") of 451 ps and 272 ps, relatively close to the 453 ps and 344 ps predicted by *Encounter*. *PrimeTime* also predicts max and min transition times of 411 ps and 217 ps, significantly larger than the 266 ps and 157 ps predicted in **clock.ctsrpt** ("Max" and "Min Rise Sink Tran"). It's possible that the differences between these estimates are due to the fact that parasitics were not extracted before *Encounter* analyzed the clock timing.

Next, check the hold-time slack in the file **timing_pt_fast_trialrouted.rpt**, and you'll see that it is -19 ps. That's not good. Our design has hold-time violations. Also, if we want to be very conservative, we may want to make this slack greater than the max transition time. The shortest path from this file is given below. Note that the insertion delay is calculated for every clock sink, which ensures that the most accurate verification of timing constraints is being performed.

```
Startpoint: u_logic/Iwp2z4_reg
            (rising edge-triggered flip-flop clocked by HCLK)
Endpoint: u_logic/Iwp2z4_reg
            (rising edge-triggered flip-flop clocked by HCLK)
Path Group: HCLK
Path Type: min

Point                                          Trans       Incr       Path
-----------------------------------------------------------------------------
clock HCLK (rise edge)                                    0.0000     0.0000
clock source latency                                      0.0000     0.0000
HCLK (in)                                      0.0000     0.0000 &   0.0000 r
HCLK__L1_I0/A (INV_X32)                        0.0013     0.0012 &   0.0012 r
HCLK__L1_I0/ZN (INV_X32)                       0.0122     0.0101 &   0.0112 f
HCLK__L2_I4/A (INV_X32)                        0.0206     0.0160 &   0.0272 f
HCLK__L2_I4/ZN (INV_X32)                       0.0357     0.0397 &   0.0669 r
u_logic/HCLK__L2_N4 (cortexm0ds_logic)         0.0000     0.0000 &   0.0669 r
u_logic/Iwp2z4_reg/CK (DFFS_X1)                0.0383     0.0099 &   0.0769 r
u_logic/Iwp2z4_reg/QN (DFFS_X1)                0.0203     0.0681 &   0.1449 f
u_logic/U261/C1 (OAI221_X1)                    0.0203     0.0001 &   0.1450 f
u_logic/U261/ZN (OAI221_X1)                    0.0310     0.0421 &   0.1872 r
u_logic/Iwp2z4_reg/D (DFFS_X1)                 0.0310     0.0000 &   0.1872 r
data arrival time                                                     0.1872

clock HCLK (rise edge)                                    0.0000     0.0000
clock source latency                                      0.0000     0.0000
HCLK (in)                                      0.0000     0.0000 &   0.0000 r
HCLK__L1_I0/A (INV_X32)                        0.0013     0.0012 &   0.0012 r
HCLK__L1_I0/ZN (INV_X32)                       0.0122     0.0101 &   0.0112 f
HCLK__L2_I4/A (INV_X32)                        0.0206     0.0160 &   0.0273 f
HCLK__L2_I4/ZN (INV_X32)                       0.0366     0.0403 &   0.0675 r
u_logic/HCLK__L2_N4 (cortexm0ds_logic)         0.0000     0.0000 &   0.0675 r
u_logic/Iwp2z4_reg/CK (DFFS_X1)                0.0392     0.0102 &   0.0777 r
library hold time                                         0.1280     0.2058
data required time                                                    0.2058
-----------------------------------------------------------------------------
data required time                                                    0.2058
data arrival time                                                    -0.1872
-----------------------------------------------------------------------------
slack (VIOLATED)                                                     -0.0186
```

V. Optimize the Design

Now it's your turn to attempt to find a better design. Modify the scripts as needed to achieve a design with the following characteristics:

- Final Detail route with no routing violations
- Clock period of at most 38 ns (i.e. at least 2 ns of slack) for the slow timing corner
- Hold-time slack greater than the maximum clock transition time for the fast timing corner

Suggested changes include the following:

- Decreasing the density of the floor-plan
- Modify the contents of the clock.ctstch file
- Selective addition of repeaters
- Modified execution of the optDesign command in any of the steps

Maintain a record of your iterations. Use the table below as a template. This table is also provided in the **iteration_record.docx** file provided with this tutorial. Note that the first iteration has been filled in for you. If one or more cells are empty for an iteration (because you forgot to record the value by accident), you will not be penalized. This is more for my information about how you went about your search and to help you keep track of what you have tried.

Iteration	1	2	3	4	5
Period	40 ns				
INIT AspectRatio	1.0				
INIT Density	95%				
CTS MaxSkew	300 ps				
CTS SinkMaxTran	400 ps				
CTS BufMaxTran	400 ps				
ROUTE type	trialrouted				
ROUTE viol/ovfl	3%H 10%V				
PT Fast Max Ins Delay	108 ps				
PT Fast Min Ins Delay	59 ps				
PT Fast Max Clk Tran	73 ps				
PT Fast Min Clk Tran	37 ps				
PT Fast Hold Slack	-19 ps				
PT Slow Setup Slack	12.6 ns				
NOTES:					

When you are done, turn in the following:

- your **CORTEXM0_routed.def** file
- your **CORTEXM0_routed.spef** file
- your **run_route.log** file, showing zero violations
- all **Tcl scripts** from your **PR** and **SYNTH** directories
- Your iteration record file, as a PDF, DOC, DOCX, or TXT file.
- your **timing_pt_fast_clock.rpt** and **timing_pt_fast_routed.rpt** files, showing sufficient hold-time slack
- your **timing_pt_slow_routed.rpt** file, showing sufficient setup-time slack

VI. Capture the Switching Activity

For the remainder of this tutorial, we're concerned with how to most accurately estimate power and the quality of the power delivery network. To achieve that accuracy, we'll need detailed switching activity from a gate-level Verilog simulation. Because our design's netlist changes throughout the flow until the *route* step, we have put this off until we have successfully routed with no violations.

The CORTEXM0 simulation that comes with this tutorial should work perfectly with the Verilog netlist generated by Encounter, you'll just need to make a few changes to point to the correct Verilog file and specify that switching activity should be captured. Do this with the following steps:

- Open a new shell window and change to the v/sim directory
- Source the setup.csh script in this directory. This script basically adds *Mentor Graphics Modelsim* and *ARM's Real View Design Studio* (RVDS) compiler to your environment.
- Add the following lines to the test-bench ("cortexm0ds_tb.v") file. These lines direct the simulator to generate a Verilog Value-Change Dump (VCD) file. This file will be very large! We will generate other switching-activity files in later steps that are more compressed, such as the Switching Activity Interchange Format (SAIF). There are ways to generate SAIF files directly, but these approaches are not very re-usable between simulators and are therefore prone to failure.

```
initial begin
  $dumpfile("waves.vcd");
  $dumpvars;
end
```

- Alter the Makefile, changing the SRCDIR variable from ../src/rtl to ../../pr. You'll also need to ensure that you remove the ../../pr/CORTEXM0DS_trailrouted.v file, if it exists, because the Makefile will attempt to analyze every Verilog file in the SRCDIR directory. Having two files that define the same module will cause an error. You'll also need to add the STDCELLSRC variable as defined below and add $(STDCELLSRC) to the list of files analyzed by the vlog

command.

```
STDCELLSRC      = /afs/eos.ncsu.edu/lockers/research/ece/wdavis/
tech/nangate/NangateOpenCellLibrary_PDKv1_2_v2008_10/verilog/Nan
gateOpenCellLibrary_PDKv1_2_v2008_10_typical_conditional.v
```

Once you've made these changes, go ahead and run the simulation to generate the VCD file with the following command:

```
make sim
```

VII. Estimate the Power

We will now estimate the power using Synopsys PrimeTime-PX. First, we'll convert the VCD file into a SAIF file, because this file is much smaller, easier to move around, and easier to archive than the VCD file. To do that, change back to your window with the PrimeTime environment settings and enter the following command:

```
vcd2saif -input waves.vcd -instance cortexmods_tb/u_cortexm0ds -output waves.saif
```

The only part of this command that requires any explanation is the *–instance* argument, which specifies the top-level module name and the hierarchical instance name of the design from the Verilog simulation used to generate the VCD files. Neither of these names is available in the routed Verilog netlist or SPEF file, but both are mentioned in the VCD file. This argument instructs the *vcd2saif* program to ignore the switching activity for other parts of the simulation when generating the SAIF file.

Next, examine the **run_ptpx.tcl** file. You should notice that it is similar to the **run_pt.tcl** script, with some notable differences:

- **power_enable_analysis and power_analysis_mode variables** – These variables have been set, instructing PrimeTime to perform power analysis.
- **read_saif** – This command reads the switching activity. It includes a *–strip_path* argument similar to the *–instance* argument used with the *vcd2saif* command.
- **report_switching_activity** – This command reports nets that have no annotated switching activity. It's important to check, because a common source of error is using the wrong SAIF file. It happens frequently, mainly because the VCD/SAIF files are so much trouble to generate.
- **report_power –** This command performs the power analysis and re-directs the output to a file.

Finally, execute the **run_ptpx.tcl** command with *pt_shell* using the command given in the README file that comes with this tutorial. If you're curious, you may want to compare this number to the value given with report_power command in *Design Compiler*, after running the *synth* step.

VIII. Analyze the Power and Ground Rails

In this section, we will use the Cadence Encounter Power System (EPS) to analyze the power rails. Unfortunately, this flow doesn't work with the 2012 version of Encounter, so you'll need to open a new shell window, change to the PR directory and set up the environment with the command:

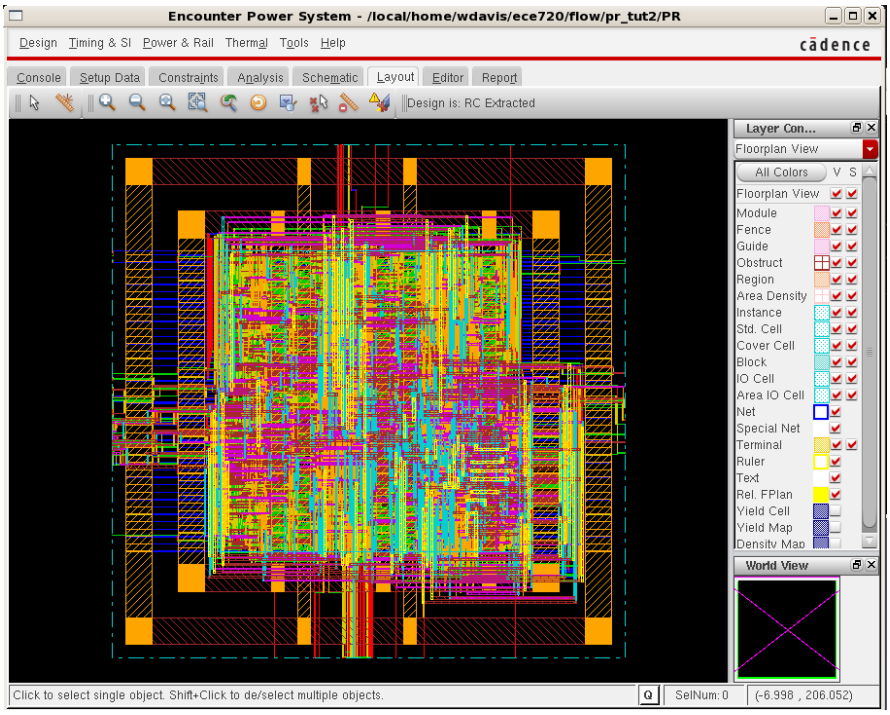
```
add cadence2010
```

Next, since you'll need to find the (x,y) coordinates of the locations on the power-rings that represent the off-chip connections to the supply (VDD) and ground (VSS) rails. These would normally be the supply and ground pads on the pad-ring, but since we have no pads in this tutorial, we will use two points on the power and ground rings. These points are specified in the the VDD.ppl and VSS.ppl files provided with this tutorial. The given file contains coordinates that will work with the floorplan initially given with this tutorial, but if you have modified the floorplan in section V above, then you will need to modify these files to run the step correctly.

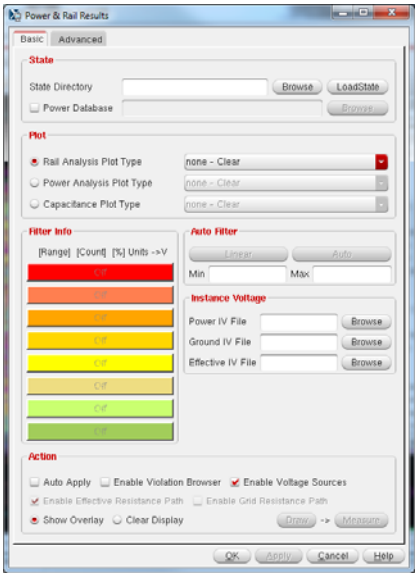
Once you have modified the VDD.ppl & VSS.ppl files, go ahead and execute the **run_eps.tcl** script, following the command in the tutorial README file. After that step completes successfully, you can use the graphical user interface (GUI) to view the results of the rail analysis, as described below.

- Start the Encounter Power System with the following Linux command:
  ```
  eps
  ```

- Choose **Design → Read Design**
  - Enter "CORTEXM0DS_routed.enc" for the design file. If you use the browser to choose this file, then you may need to set the "Files of Type" select box to "Design Files (*.enc)".
  - Ensure that "Read Physical Data" is checked
  - Click OK
  - After completion of this step, you should be able to click the "layout" tab in EPS and view the layout, just as you do within the Encounter tool, as shown below.



- Choose Power & Rail → Report → Power & Rail Results…
  - You should see the Power & Rail results dialog, as shown below



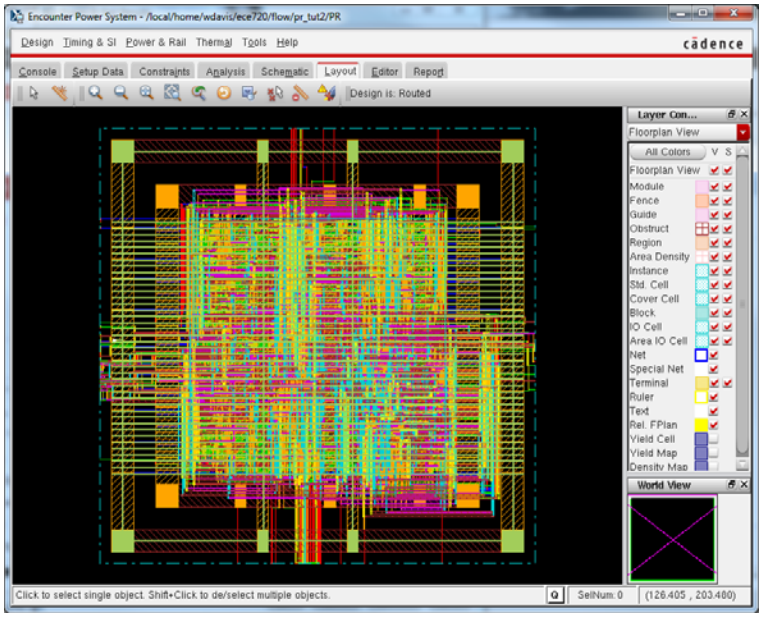
Before viewing the Voltage storm rail analysis results, you'll need to load the voltage storm "state directory". There are two state directories generated, one for VDD and one for VSS. Load the VDD state with the following steps:

- Next to the "State Directory" box in the Power & Rail Results window, click "Browse"
- Browse to the folder PDCore_25C_avg_1/VDD
- Click Choose (be sure not to choose a subdirectory within the directory mentioned above)
- Next to the "State Directory" box in the Power & Rail Results window, click "LoadState"

Once the state is loaded, you should be able to select the "Rail Analysis Plot Type" button in the Power & Rail Results window, and then select various results. The options will be grayed-out before the state is loaded.

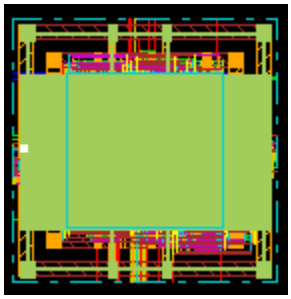
To view IR drop, follow these steps:

- Select the "Rail Analysis Plot Type" button in the Power & Rail Results window, under "Plot"
- Select "ir – IR Drop" in the select-box next to this button
- Click Apply
- You should see the layout updated as shown below:

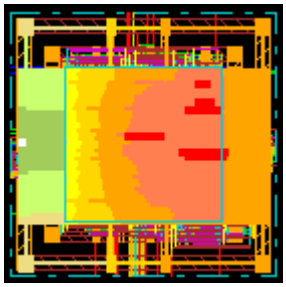


- These results show the IR drop color-coded to follow the legend in the Power & Rail Results window. It's a little hard to decipher, since the colors are superimposed over an already colorful layout. To make things easier to see, you may want to zoom out by hitting shift-Z. Do that, and

you will see the following:

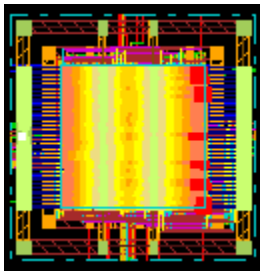


- At first glance, the image above shows that no analysis has been done, but actually the IR-Drop is simply too small to see. To fix this, we need to change the legend to match the actual simulation values. Do this by clicking the "Auto" button, under the "Auto Filter" heading. Then click "Apply" again. You should see the legend updated and the layout updated as shown below:

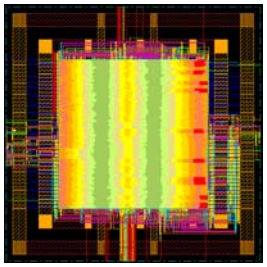


Using this approach, you can also view other metrics of the rail, as computed by VoltageStorm:

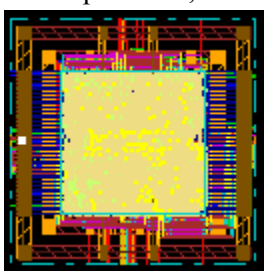
- **res** – Grid Resistance, effective resistance to the pad for every point on the rail



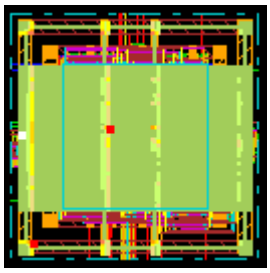
- **reff** – Instance Effective Resistance, same as Grid Resistance (res), except that it it printed for each instance, rather than each point on the rail.



- **tc** – Tap current, the current drawn for each port (tap) connected to the rail



- **er** – Electromigration Risk, given as the reciprocal of the Mean Time to Failure (MTTF) in hours

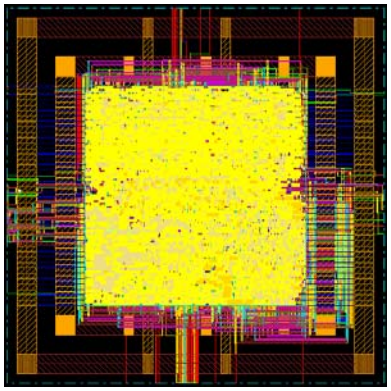


In addition to Voltage Storm results, you may also want to view the output of the power analysis results, to help interpret the source of various problems. To do this, you will need to load the power database, using the following steps:

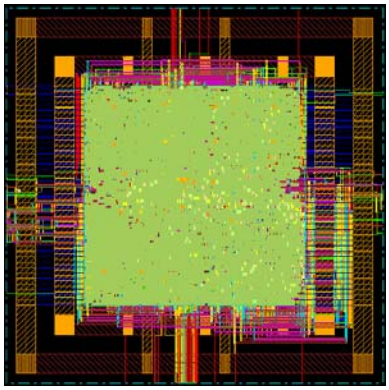
- Check the "Power Database" check-box in the Power & Rail Results window, click "Browse"
- Browse to the file "static_power_max/power.db"
- Click "Open"

Once the power database is loaded, you should be able to select the "Power Analysis Plot Type" button in the Power & Rail Results window, and then select various results. The options will be grayed-out before the power database is loaded. Two of my favorites are listed below:

- **ip** – Instance total power, the total power dissipated on each instance



- **ipd_s** – Instance Switching Power Density, the power density for each instance (in $mW/\mu m^2$) due to switching alone (no leakage or internal power)



For more information, the most helpful documents for understanding the EPS tool are the following:

- **Encounter Power System User Guide**
  /afs/eos.ncsu.edu/dist/cadence/.install/lnx86/ets91/doc/epsUG/epsUG.pdf
- **Encounter Power System Text Command Reference**
  /afs/eos.ncsu.edu/dist/cadence/.install/lnx86/ets91/doc/epstxtcmdref/epstxtcmdref.pdf

Also, because EPS is a wrapper for other tools, and because Place & Route Tutorial #3 focuses on the cell-based analysis using Voltage Storm PE, the documentation for that tool may also be helpful to gain insight into what the tool can do.

- **VoltageStorm Cell-Level Rail Analysis User Guide**
  /afs/eos.ncsu.edu/dist/cadence/.install/lnx86/ets91/doc/vstormpe_ug/vstormpe_ug.pdf
- **VoltageStorm Cell-Level Rail Analysis Reference**
  /afs/eos.ncsu.edu/dist/cadence/.install/lnx86/ets91/doc/vstormpe_ref/vstormpe_ref.pdf