Generating accurate gate level activity for power analysis, prior to back annotated timing simulation

Carsten Rau (Infineon Technologies AG)

Infineon Technologies AG

Neubiberg, Germany

ABSTRACT

Accurate Power or Voltage Drop Analysis on Gate level requires activity data, which should also be generated on Gate Level, though this is usually only available at a very late stage. Using RTL activity data on the other hand may be dangerous for designs with large high active combinatoric clouds.

This paper will discuss a method to generate accurate gate activity data with only RTL activity information available as input, thus enabling early analysis and a true "shift left".

Table of Contents

Table of Figures

1. Introduction

1.1 Motivation

This paper was actually triggered due to some findings during application of SpyGlass power estimation on RTL described in another SNUG paper for Germany 2016 [1].
The design was a digital controller for power management application in a 65nm mode.

During first gate level power analysis of the design, we found that some modules had a much larger than expected deviation from our RTL estimation. After some investigations, we found out that the root cause is a mismatch in activity data. The affected modules had an up to 10X higher activity on GATE than on RTL. This was due to glitches on combinatoric logic.
While searching for ideas how to find out earlier if such an issue exists, we found a SNUG paper [2] describing the Siloti flow and decided that this approach might be beneficial enough to apply this on that design for further analysis.
Why beneficial?

- It may be an approach to earlier see if there is a severe glitch related activity mismatch
- It may be an approach that avoids vendor specific internal RTL activity annotation algorithms and therefore have common activity data for all tools
- It may be an approach to get valid GATE activity prior to timing clean gate level simulation

1.2 Traditional Power estimation with RTL activity

Before elaborating on the new approach, let's revisit the traditional approach.
If for some reason one does not have yet matching gate level simulation results for the gate level design, basically every EDA power estimation tool offers options to use RTL activity instead.
This will then be propagated internally through the GATE design, so that each net gets activity annotated. Such a propagation algorithm may be something similar as described in

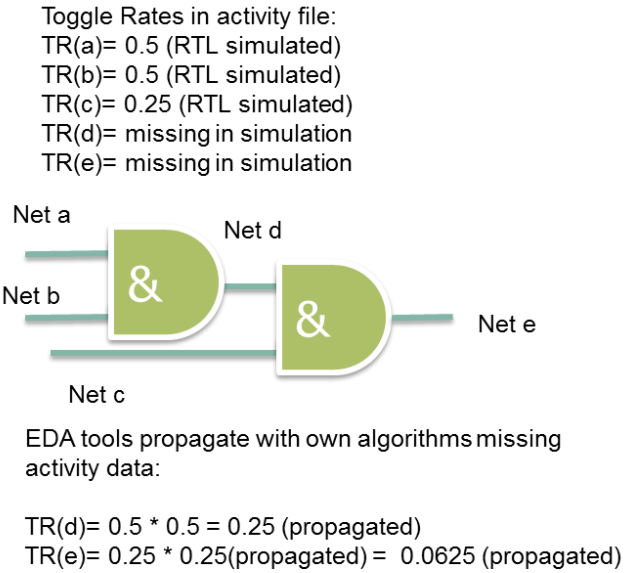
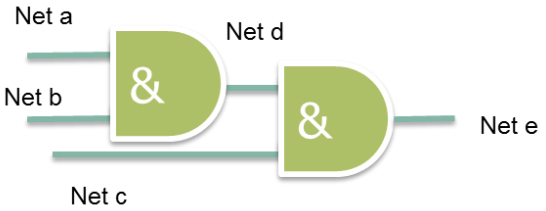


Figure 1:

Toggle Rates in activity file:
TR(a)= 0.5 (RTL simulated)
TR(b)= 0.5 (RTL simulated)
TR(c)= 0.25 (RTL simulated)
TR(d)= missing in simulation
TR(e)= missing in simulation



EDA tools propagate with own algorithms missing activity data:

TR(d)= 0.5 * 0.5 = 0.25 (propagated)
TR(e)= 0.25 * 0.25(propagated) = 0.0625 (propagated)

Figure 1 activity propagation example

This is just an example. The actual algorithm might be a bit more sophisticated, but fact is:

- It is not the exact same propagation algorithm in all EDA tools.
- The more original simulated net activities are missing the more often the propagation algorithm is triggered and the less accurate the overall activity is
- When propagation is triggered for a fanout cone of an original simulated net, the higher its toggle activity and the more combinatoric nets in the fanout of that net have to be calculated by the propagation engine, the higher the error in the propagated activity.

So if you want to:

- Correlate EDA tools
- Debug Activity Data
- Get accurate activity for high active deep combinatoric paths which are prone to glitches due to e.g. race conditions

You will not be able to properly do this based on default RTL activity propagation approach.
What we are looking for is a solution that:

- Is doing the activity propagation for all tools to allow correlation and common debug by generating a new activity file for all EDA tools
- Is able to handle glitches and deep combinatoric paths in its activity propagation algorithm
- Can be applied before having a timing clean full functional gate level simulation running

The described flow fulfills those requirements.

2. Activity Generation Flow

2.1 General Flow

The details of the Verdi/Siloti/Wisim flow have already been described in the SNUG paper [2] and is also described in Synopsys Online Documentation: Siloti User Guide and Tutorial [3], as well as Synopsys Online Documentation: Siloti Replay Simulation User Guide[4].
Hence I give here only a short overview:

The flow consists of 3 major steps, as shown in Figure 2:

- 1. “Generate STA data” (with Primetime)
  - Generate SDF
  - Run a primetime tcl script to generate clock root to Q pin delay for all registers
  - Goal: To allow realistic delay of register activity as clock tree does not exist on RTL
  - Output: delay map file: “<register> #<clock root->Q delay>“
- 2. “correlate GATE to RTL”
  - Read RTL + Gate Design into Verdi and run crdb
  - Goal: Create for each register, primary input, black box output in the Gate level design a signal mapping to its RTL equivalent
  - Output: signal mapping file: “<GATE signal> => <RTL signal>“
  - This step is currently the most complex part of the flow
- 3. “run simulation”
  - Verdi “cockpits” digital simulator via PLI interface
  - Goal: Simulate gate level design by forcing all signals in mapping file, whenever RTL equivalent signal toggles in RTL fsdb file => no need for timing clean design!
  - Time of force = RTL toggle time + delay from delay map
  - Output : new pseudo-gate fsdb file (containing new activity to be used by any other Tool)

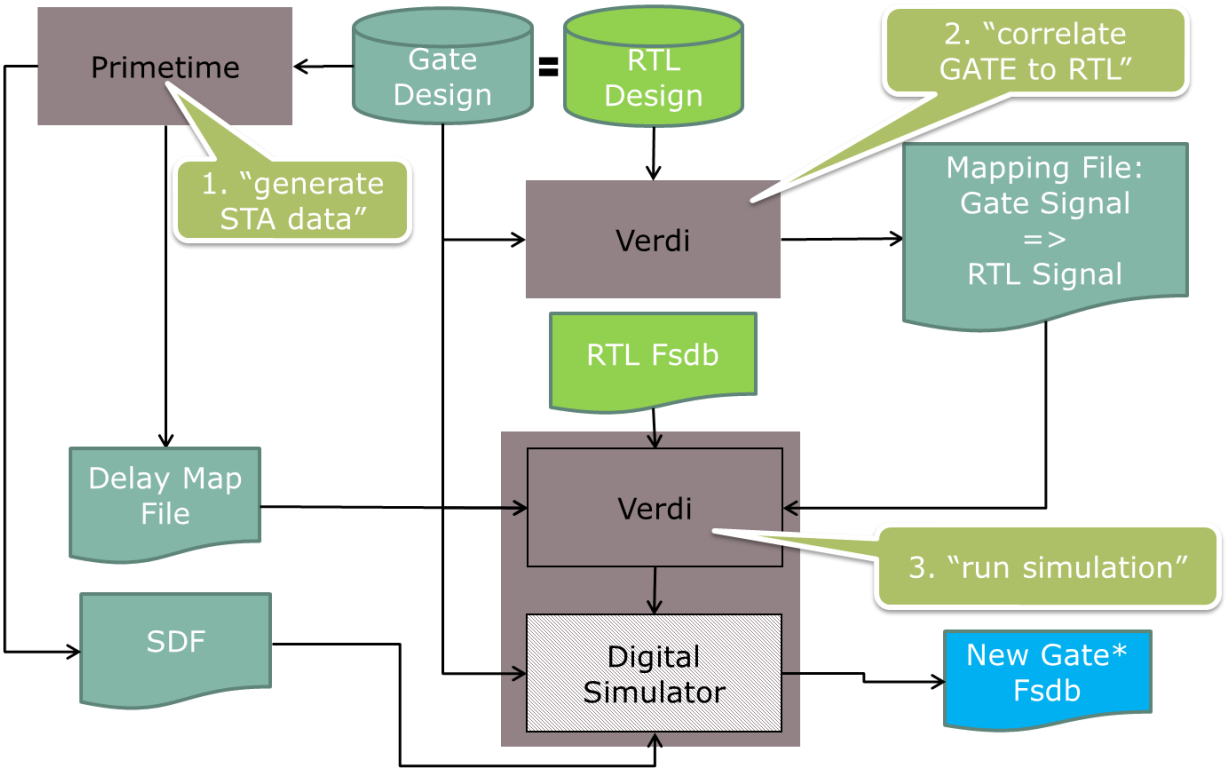


Figure 2 Verdi/Siloti/Wisim flow chart

So overall this flow is simply a new activity file generator or "propagation engine", which can be applied on basically all levels. Depending on quality of inputs, the generated activity correlates to final tapeout layout simulation activity data more or less.
So simply as closer one gets to the actual tapeout data, the more accurate also the output of this flow becomes.
One could run this with a synthesis netlist and a SDF generated based on wireload models or one could run this on the final tapeout netlist with a SPEF based generated SDF.
The Figure 2 above describes the basic flow, but if you use as the "Gate Design" a synthesis netlist or a layout netlist is up to the user. Obviously like in any other design step, the closer you get to the final database, the more valid its results.

The most complex and important step is step 2, the mapping step. It is indeed a similar task to equivalent checking, but here we just want to map dedicated signals and even use the naming convention for the RTL signals as used by the RTL simulator, which might be different than in other tools (e.g. generate loop naming convention). For that step it is possible to read also a SVF files to handle hierarchical name changes done during synthesis as well as a name rule file, which can cover some general as well as instance specific name change rules.

In step 3 then Verdi provides the automated environment to watch all signals in the mapping file in the fsdb and then applying a force on respective signal in the gate level netlist and thus effectively mimicking a real gate level simulation for the design.
Currently the digital simulators VCS and Ncsim are supported as "propagation engine".

2.2 Used Flow for this Paper

As this was a new investigation, we first wanted to see if the concept in general works.
So we applied the concept at the same layout database, on which we also executed a real simulation.
So we generated the "pseudo gate" fsdb with the described flow using:

- Same layout netlist as used for real layout simulation
- Same SPEF based SDF as used for real layout simulation
- Delay map file generated with custom primetime tcl script sourced during SDF generation
- RTL FSDB file that contains same use case as real layout simulation
- For mapping we used custom Perl script to fix the mapping, as SVF files were not available and hierarchical name changes for the given design were limited
- Third party digital simulator as "propagation engine"

Just a quick look at the basic command structure and not elaborated further as really already very good described in referenced paper and online documentation:

```
#generate mapping:
crdb -allreg -expGate -expRTL -top_inport -expGateUncor \
-RTL "-top rtl.top -nclib" \
-GATE " -top gate.top -nclib " \
-crdb corr.crdb -expMap corr.map -excludeModule bb.f


#prepare and run simulation:
wisim -lib gate -top top -config config_file -replay_extract
wisim -lib gate -top top -config config_file -replay_compile
wisim -lib gate -top top -config config_file -replay_sim
# where config_file contains pointers to input data
# and 3third party compile and run script
```

Overall after we learned how to use the flow, primary work was the tuning of the mapping, as it is important to really achieve 100% mapping of the intended signals.
So while rule of thumb is that with 95% mapping rate one can assume good result, we found that if you are missing by chance in those last 5% some configuration signals which enable or block important clock gates, it may have significant effect on clock tree activity data.

Meanwhile some enhancements should be available in the tool like only dumping out output pins of black box hard macros or not including latches when dumping all register signals.
But for this paper those enhancements were not available, so custom workarounds were used for that purpose.

3. Results

This section provides some results and comparisons achieved with the described flow.
The difference to earlier mentioned SNUG paper [2] is that it provides also some results on its effect on lower hierarchies and net activity.

The following pages basically contain four types of plots, which I would like to quickly explain:

- Activity scatter plot
- Activity histogram
- Power scatter plot
- Power histogram

Activity scatter plot:

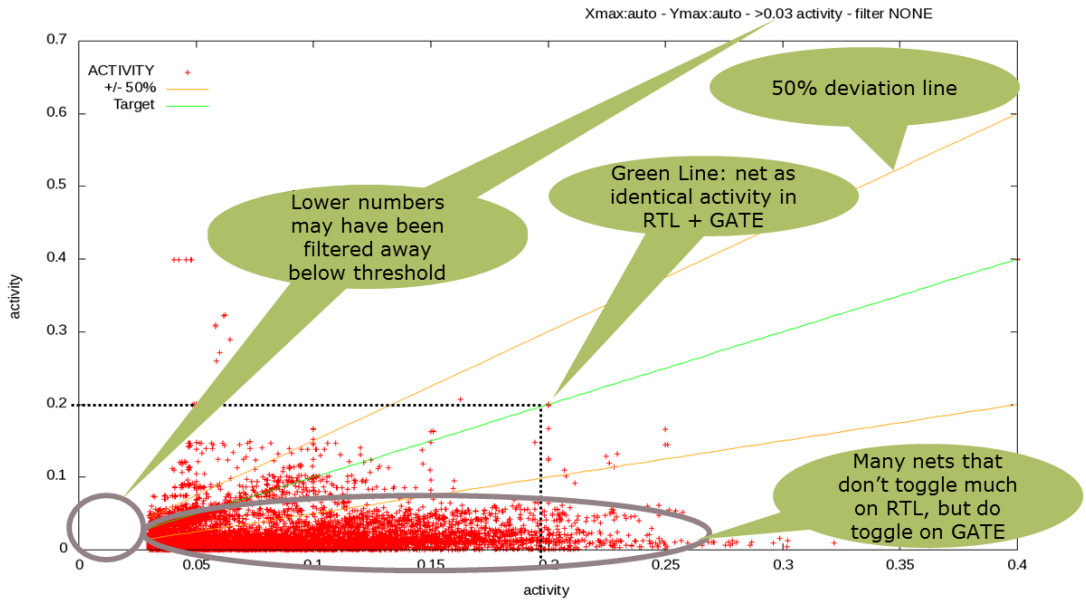


Figure 3 activity scatter plot example

In the activity scatter plot the X and Y axes show the average number of toggles per clock cycle.
So the highest possible number should be 2, which would be the clock signal itself.
On the bottom we will have the reference (X) usually the data based on the real life gate level simulation, while on the left hand side (Y) we have the compare data, which is usually based on using the original fsdb or generated "pseudo gate" fsdb file.
The red data points represent the activity of a given net using reference as well as compare run.
So if all the data points are on the green line, the results are identical.
Some graphs might have a blank spot in the lower left corner. This is simply due to the fact that low activities were filtered away as anyway not that relevant for power, but would ruin the fitting histogram, due to outliers being dominant for very small numbers.

Activity histogram:

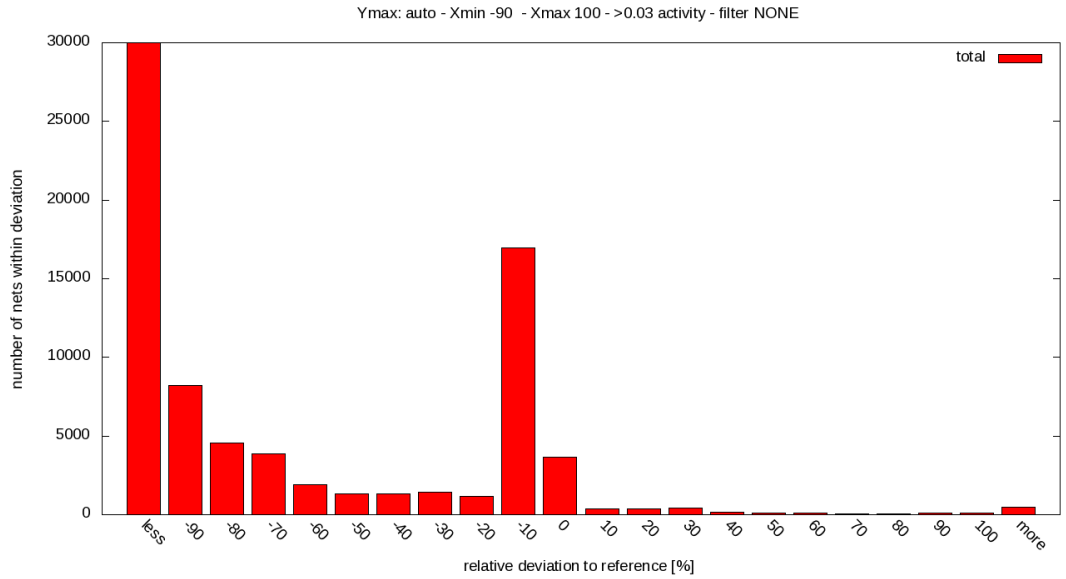


Figure 4 activity histogram example

The histogram simply represents the same data in a different view. Instead of working with the actual numbers, it shows the deviation in percent for each of the data points.

Power activity plot and histogram:

Similar graphs are generated also for total (red), switching (green) and internal (blue) power, only lacking the scale for absolute numbers.
For the histogram the different types of power are stacked on top of each other, and for easy histogram read I recommend simply focusing on the red total power comparison.

3.1 PrimetimePX

First we start with a comparison within PrimetimePX for average power estimation.
So we once ran Primetime PX with the original RTL activity and once with the generated pseudo activity on a layout design state and compared both with the results from the real gate level simulation activity file.
Figure 5 and Figure 6 show comparison with original RTL activity file.

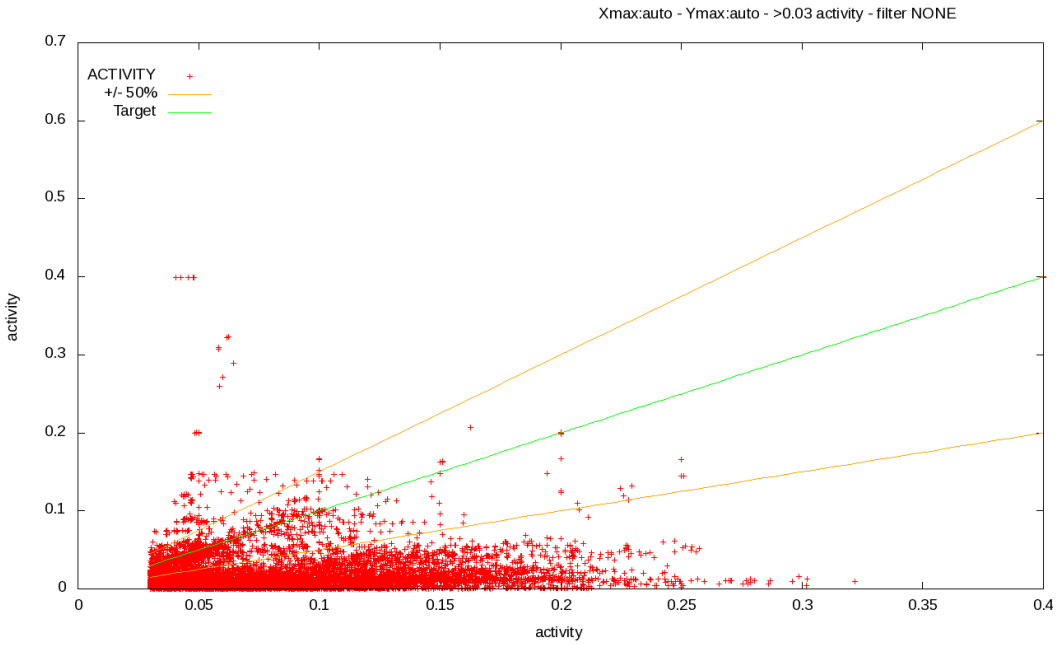


Figure 5 PrimetimePX ORG RTL activity scatter plot

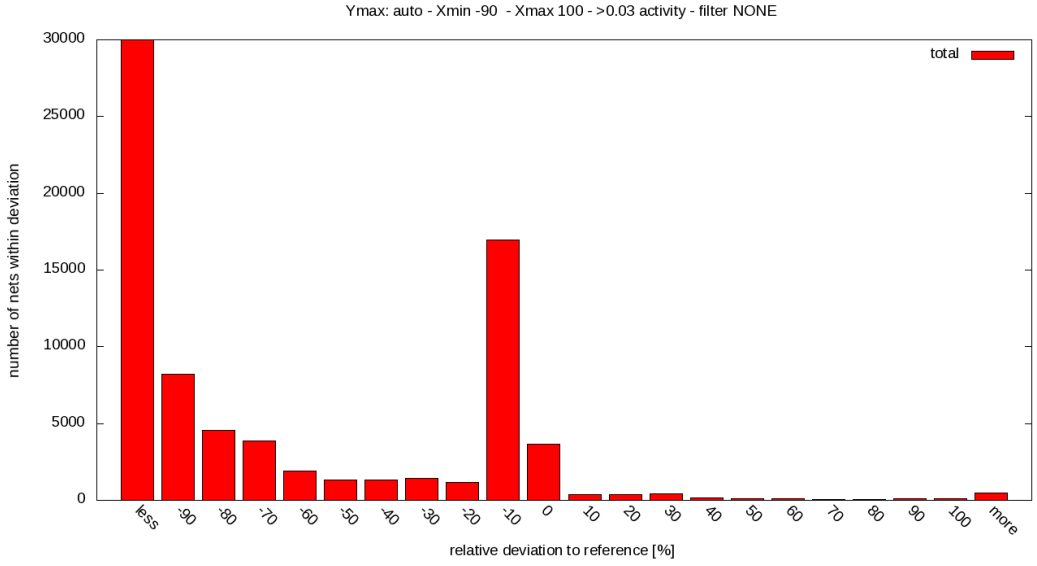


Figure 6 PrimetimePX ORG RTL activity histogram

Scatter plot Figure 5 as well as histogram Figure 6 show that for the given design and use case a significant difference in activity exists. Basically all nets show much less activity if simply relying on the internal activity propagation of the tool by using the RTL activity file.

Such a mismatch in activity also results in a mismatch in power estimation as visible in scatter plot
Now we compare our Siloti/Wisim generated “pseudo gate” activity file with the standard gate level and take a look at the similar graphs:

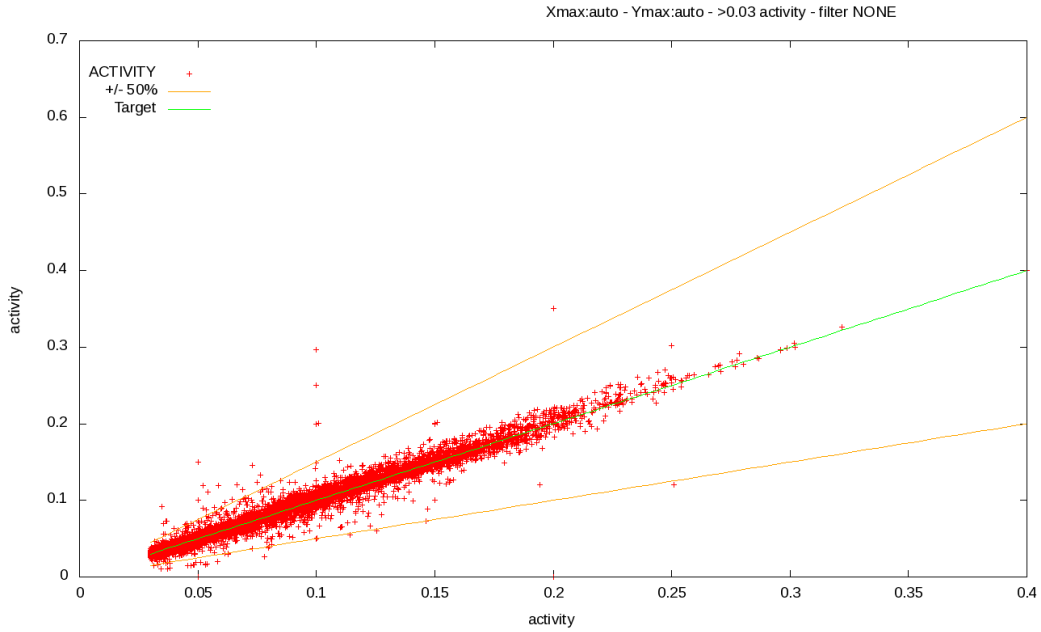


Figure 9 and histogram

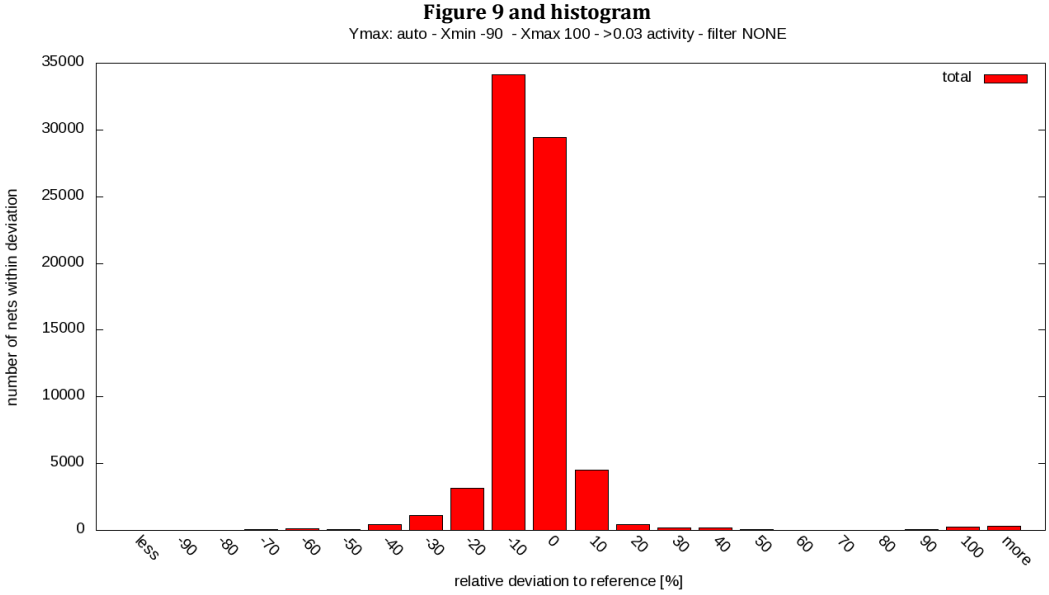


Figure 10

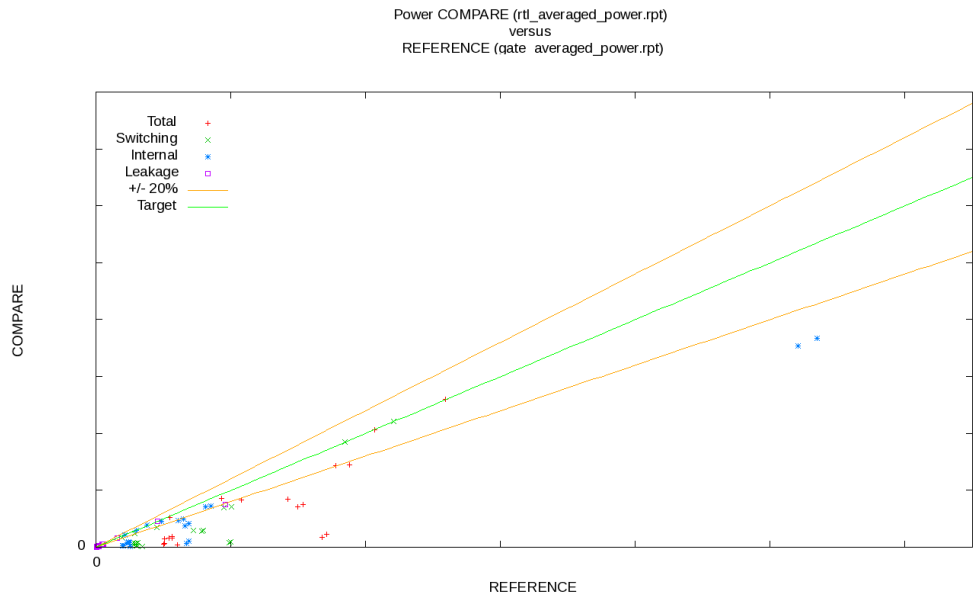


Figure 7 PrimetimePX ORG RTL power scatter plot

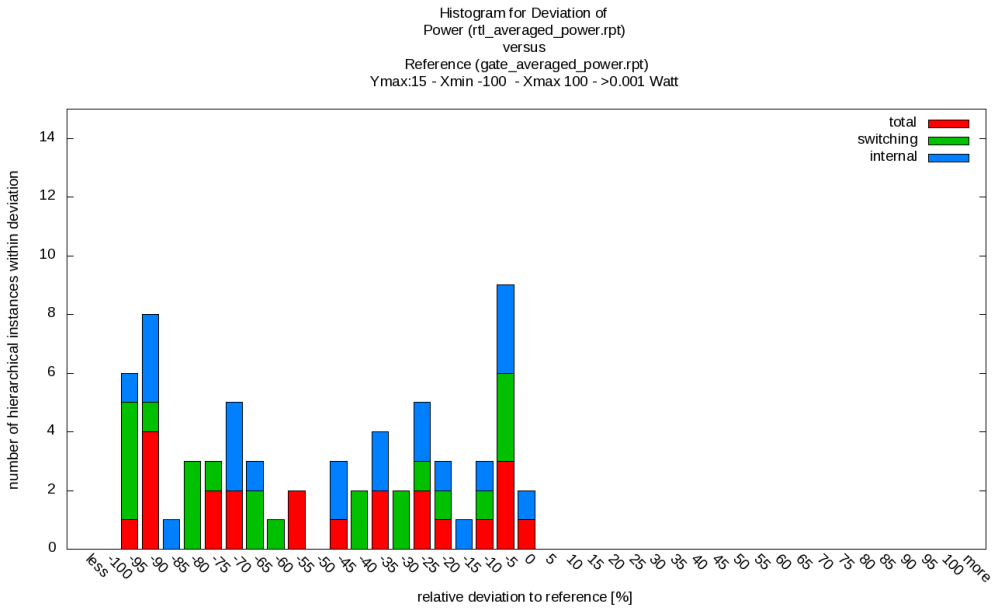


Figure 8 PrimetimePX ORG RTL power histogram

In the above two figures now power values of design hierarchies are compared.
The scale is missing in the power scatter plot files, but that does not hinder us from recognizing that all data points are on the lower side, hence showing much lower power consumption using RTL activity
In scatter plot and histogram one can see those modules consume less power using the RTL activity file than the original GATE activity file, which is the reference.
Remember that the only difference in input is RTL or GATE activity, all other inputs are identical.

Now we compare our Siloti/Wisim generated “pseudo gate” activity file with the standard gate level and take a look at the similar graphs:

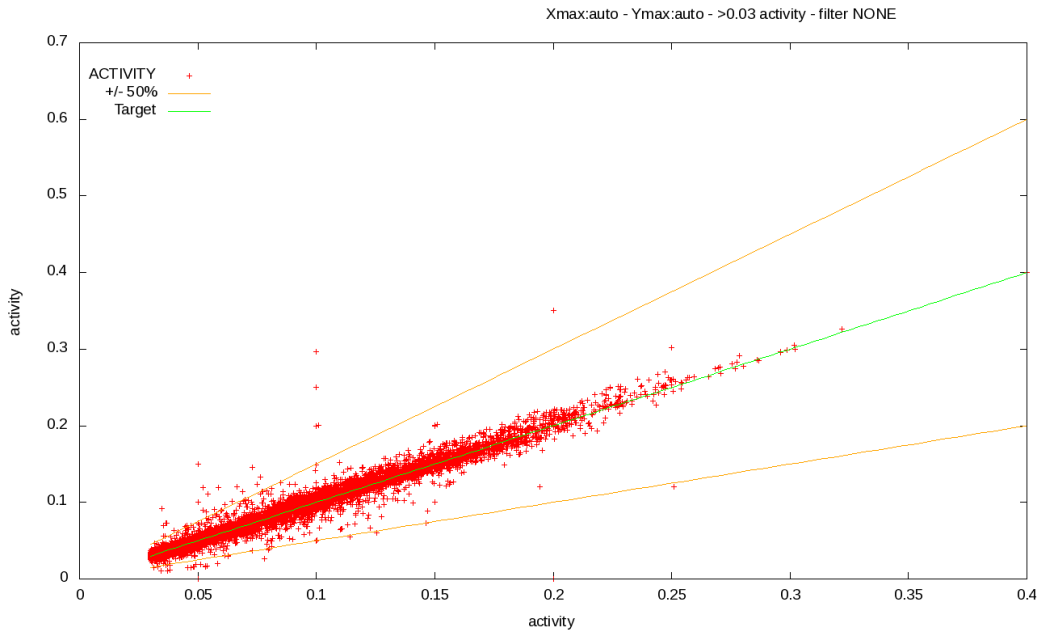


Figure 9 PrimetimePX WISIM GATE activity scatterplot

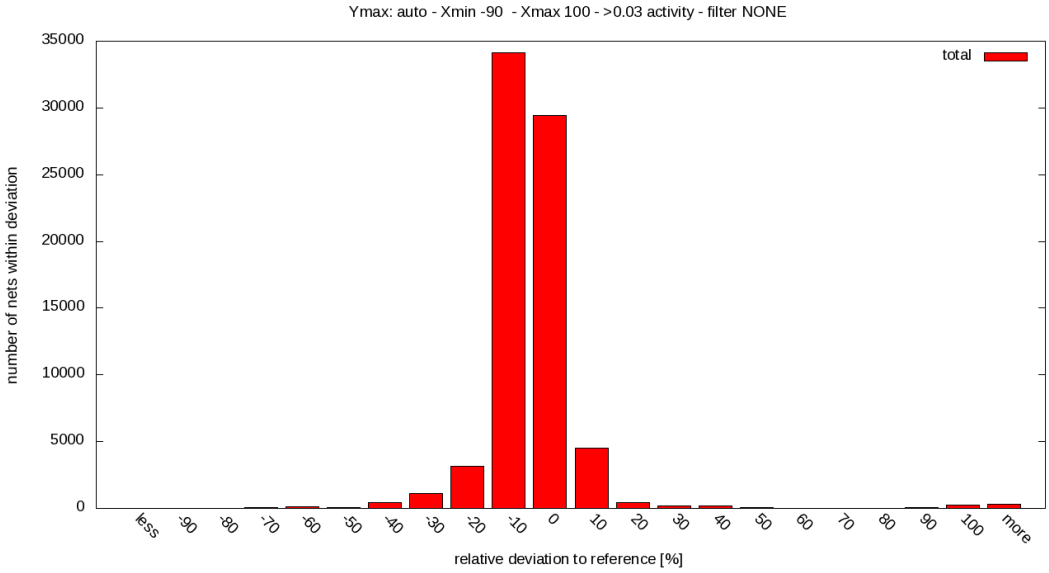


Figure 10 PrimetimePX WISIM GATE activity histogram

In Figure 9 and Figure 10 we can see that all the individual average net activity again is much closer together. With those net activities this should also result in much better power correlation, which indeed is visible in **Error! Reference source not found.** and **Error! Reference source not found.**

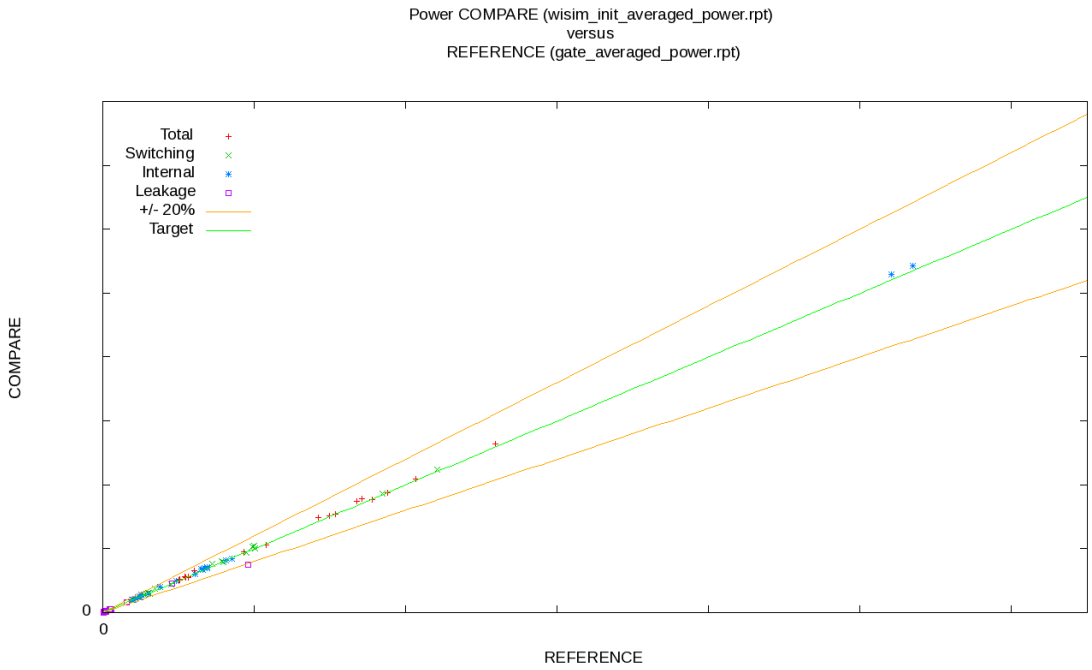


Figure 11 PrimetimePX WISIM GATE power scatter plot

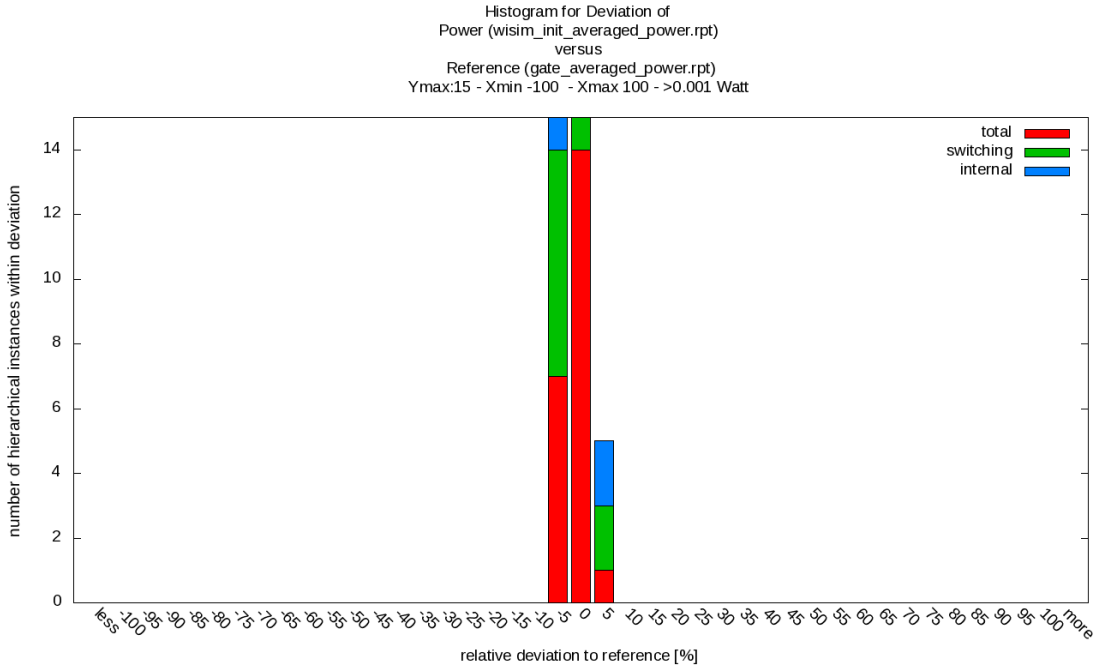


Figure 12 PrimetimePX WISIM GATE power histogram

Here we see now that basically all design hierarchies show almost identical power consumption. Indeed the somewhat spread net activity averages out and results in very good correlation for all module power.
So the information that this not just averages out on top level is also very assuring of the concept.

3.2 SpyGlass PE

Normally SpyGlass PE is not used on gate level design analysis, but as in this design we also had some RTL SpyGlass PE analysis, we wanted to confirm that the mismatch to GATE is mostly due to the activity data.
Hence for this evaluation and investigation also SpyGlass PE was run on the final layout database, using once the RTL activity , once the real layout simulation activity file and once the "pseudo gate" generated activity file and we are able to create some graphs very similar to PrimetimePX.
Only the activity graphs are explicitly shown in this paper as the overall result is very similar to PrimetimePX.

Figure 13 SpyGlass PE ORG RTL activity scatter plot

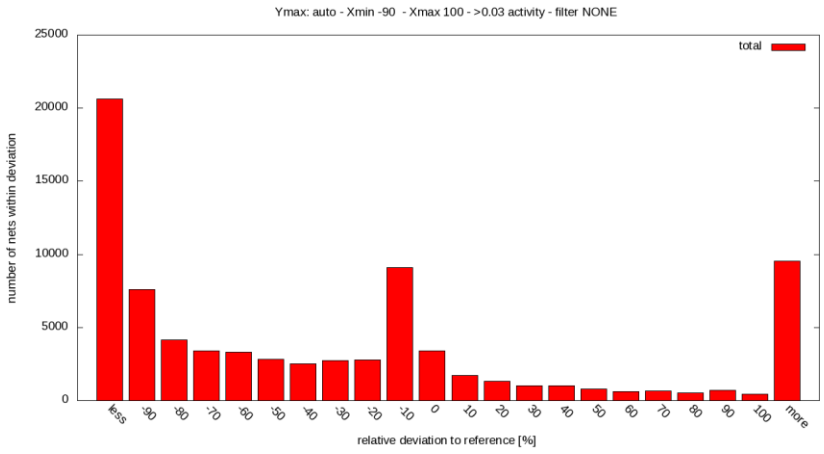


Figure 14 SpyGlass PE ORG RTL activity histogram

Again we see the significant difference in activity when looking at

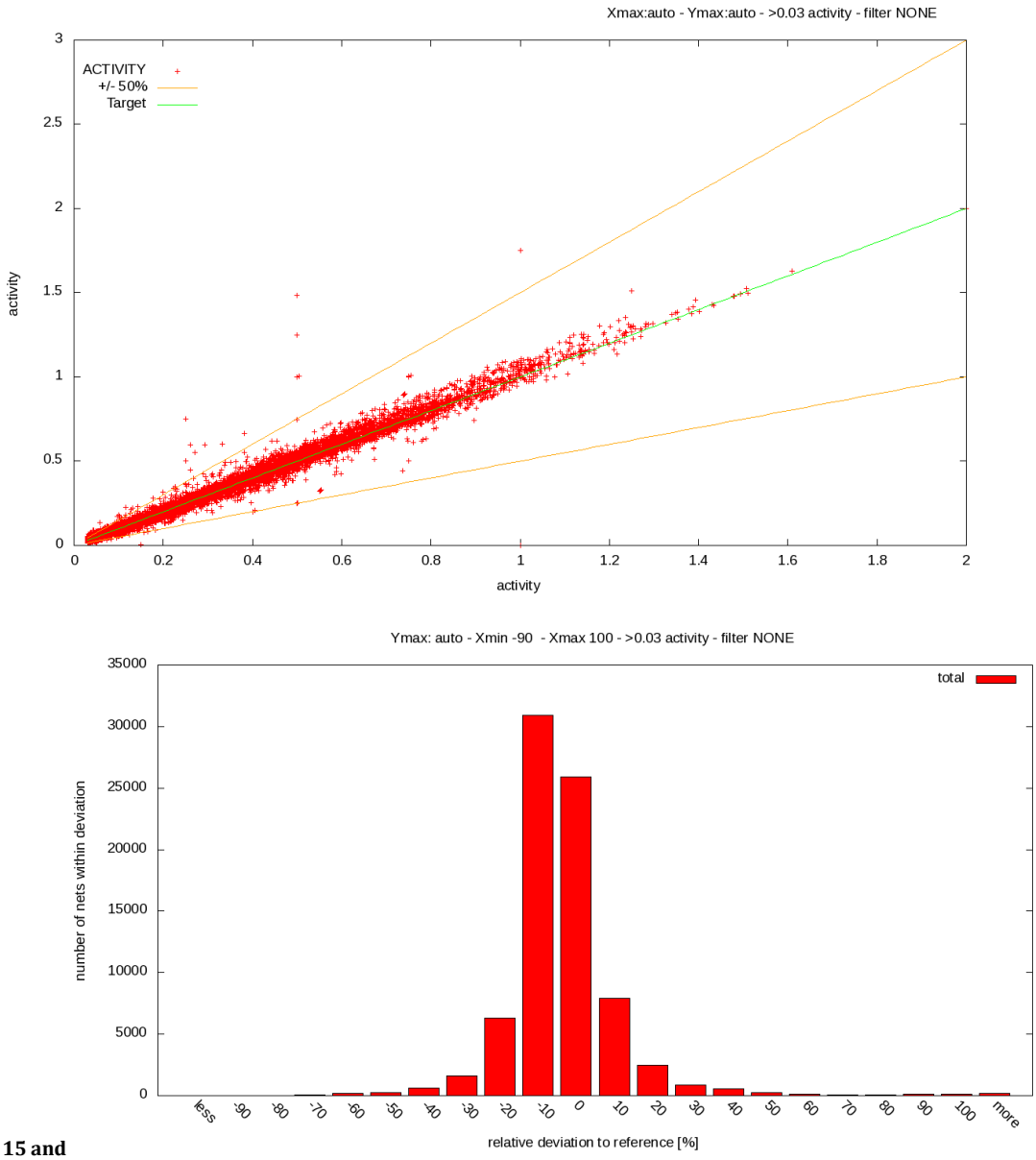


Figure 15 and

Figure 16.

PrimetimePX seems to have here the better RTL activity propagation algorithm, compared to SpyGlass PE. The activity graph of SpyGlass PE differs far more often when using the original RTL activity than PrimetimePX and a lot more outliers are visible which actually have higher activity then on gate level. This is just another proof that for EDA feature debugging, it is very convenient to have an independent activity generator instead of debugging tool internal propagation engines.

Comparing now again the "pseudo gate" generated activity with the real gate simulation data, we see the similar picture as in PrimetimePX and the correlation improved significantly.
All the net activities are much closer to the reference.

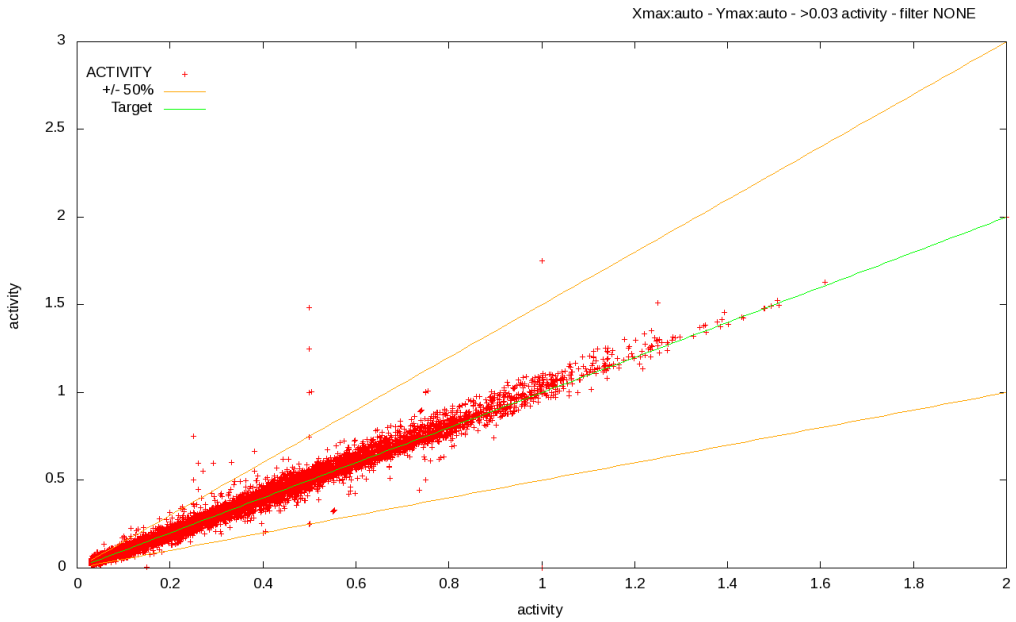


Figure 15 SpyGlass PE WISIM GATE activity scatterplot

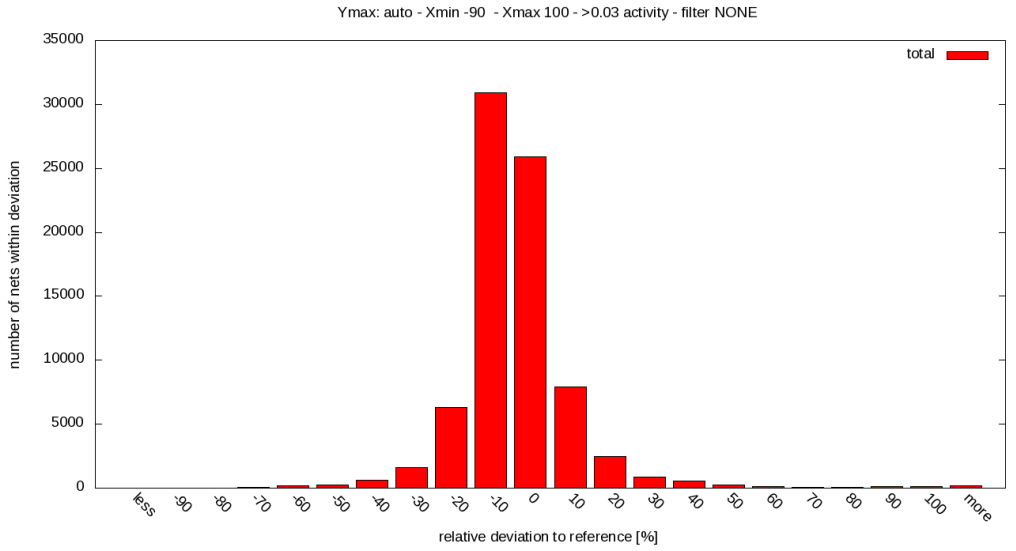


Figure 16 SpyGlass PE WISIM GATE activity histogram

3.3 Voltage Drop

Respective different activity data files were also put in a third party voltage drop analysis tool. This is an application that will also profit from accurate activity prior to timing clean layout simulation. The time window of the used activity files was just shortened for this voltage drop analysis and only average activity was used for generating the voltage maps (=static voltage drop based on VCD).
Again we fed the tool once with the original RTL activity, once with the „pseudo gate“ activity and once with the respective real simulation data.
You can look at the layout voltage maps below at **Error! Reference source not found.**, **Error! Reference source not found.** and **Error! Reference source not found.**.

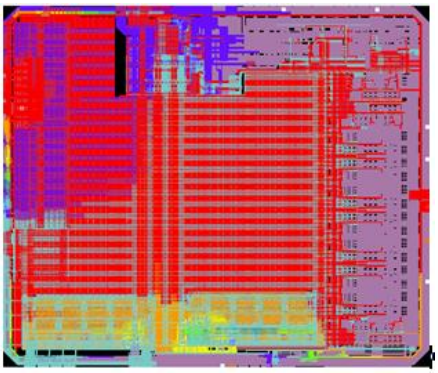


Figure 17 Voltage map ORG RTL

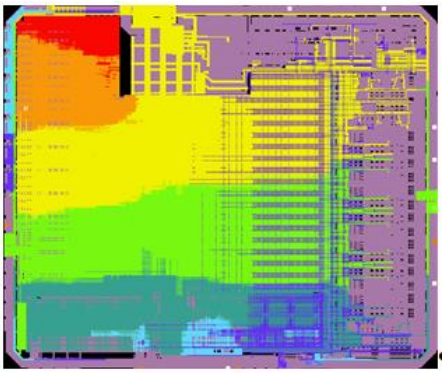


Figure 18 Voltage map WISIM GATE

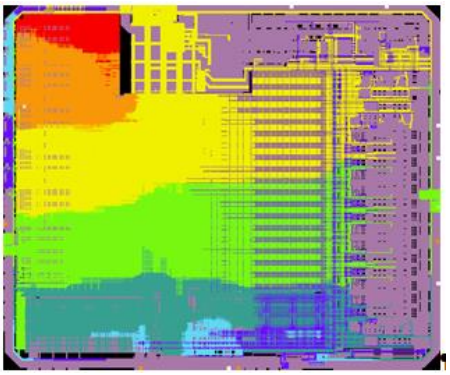


Figure 19 Voltage map ORG GATE

The color maps were purposefully not synchronized, so for each voltage map picture, the tool picked the color by its own, spreading its color code between the max and min voltage drop value encountered. So even though completely red, the actually voltage drop using the RTL activity is much lower than those encountered with the real or pseudo gate activity.
And as clearly visible the "pseudo gate" picture maps to the real gate almost completely.
There are just very few minor areas (check e.g. top left corner) where one can see that it is not actually just the same picture put here twice.

4. Conclusion

This paper showed that for average power and voltage analysis the approach works and produces from power perspective valid trustworthy activity data. So not just top level numbers fit, but also the distributions on lower design hierarchies do not show significant outliers.
This could be used to get earlier in the design flow indication about power or voltage drop or identify modules that have a significant delta between RTL and GATE activity, so in other words are prone to glitches/race conditions.

While the flow is working, especially in the area of the actual RTL to GATE mapping file generation, this is an area for improvement. Here some custom tuning is needed to handle specific naming convention or flattening or reordering of design hierarchies. Some enhancements have been made to the tool meanwhile and also using SVF information will help in that mapping step.

5. Acknowledgements

Tim Braeuninger (Infineon Technologies AG)

Soenke Grimpen (Infineon Technologies AG)

Jens Dickel (Synopsys)

6. References

[1] SNUG Germany 2016 paper: Quo Vadis Power? Early Power Assessment over Development Cycle

[2] SNUG Silicon Valley 2014 paper: Early Vector Based Dynamic Power Estimation (tc06_parigi_paper.pdf)

[3] Synopsys Online Documentation: Siloti User Guide and Tutorial

[4] Synopsys Online Documentation: Siloti Replay Simulation User Guide