Early power analysis flow using RTLA and RTL-PrimePower, with a focus on glitch power

Nicola Imperato, Felice Tecce

STMicroelectronics
Napoli Italy

www.st.com

ABSTRACT

A Frontend kit customization for early power analysis has been set up using RTL-Architect (RTL-A) and RTL-PrimePower (RTL-PP) tools.

Exploiting the capability of RTLA synthesis engine is possible to estimate clock network power before physical implementation starts, while with RTL-PrimePower is possible to use RTL simulation waveforms to analyze the main functional modes without setup gate level simulation.

A focus was put first on clock network power analysis, finding the best clock gating strategy, then on glitch power analysis.

Glitch power can be a significant part of power because of long arithmetic paths in DSP, so exploiting the capability of RTL-A to cross-probe and rank the glitch power in the RTL file, it is possible to optimize the power selecting low glitching arithmetic design architecture for synthesis.

Table of Contents

1. Introduction 3
1.1 CMOS power consumption 3
1.2 Glitches power consumption 5
2. Prelayout power estimation 6
3. Prelayout power analysis flow using RTL-A and RTL-PP 9
4. Power analysis result 13
4.1 Glitch power optimization with RTL-A 17
5. Conclusions 19
6. Acknowledgments 19
7. References 19

Table of Figures

Figure 1 - Activity Switching in CMOS 4
Figure 2 - Short-circuit current in CMOS 4
Figure 3 - Leakage current in CMOS 5
Figure 4 - Logic example for Glitches 6
Figure 5 - Waveforms Glitches example 6
Figure 6 Postlayout power analysis. 6
Figure 7 Fronted kit with RTLA- RTL_PP 9
Figure 8 Hybrid RTL-A RTL-PP flow 10
Figure 9 RTL-A basic script. 11
Figure 10 RTL Prime power script. 11
Figure 11 Glitch power setup 12
Figure 12 Glitches with shifted delay analysis 12
Figure 13 report glitch power metric by file 15
Figure 14 Glitch power RTL file cross probe 16
Figure 15 Synthesis directives. 17

1. Introduction

In a world where electronics require increasing miniaturization and performance enhancement, attention must be paid to power consumption.

Power consumption represents the energy that is mostly dissipated as heat during the operation of electronic circuits. In the case of CMOS circuits, power consumption depends mainly on the supply voltage, clock frequency, and transistor size.

The instantaneous power dissipated by a CMOS circuit can be calculated as the product of the supply voltage and the current flowing through the circuit at a given time. In formulas:

$$P(t) = V(t) * I(t)$$

where P(t) represents the instantaneous power, V(t) the supply voltage, and I(t) the current flowing through the circuit at time t.

The total power dissipated by a CMOS circuit can be calculated as the integral of the instantaneous power over a period of time T. In formulas:

$$P_{Tot} = \frac{1}{T}\int_0^T P(t)dt$$

where P_{Tot} represents the total power dissipated by the circuit over a T.

The analysis of power consumption is of great importance for the design of energy-efficient CMOS circuits, as it allows the evaluation of the impact of design choices on energy dissipation and the identification of any critical points that require optimization.

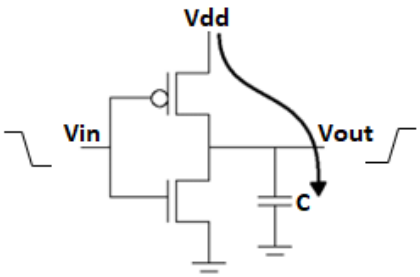
There are several factors that generate power dissipation (both static and dynamic) in a CMOS and in particular attention is focused on:

- Switching activity
- Short Circuits Currents
- Leakage Currents
- Glitches

1.1 CMOS power consumption

The term switching activity refers to the change in the logical state of a given node from logic 0 to logic 1 or vice versa. Switching activity is an important parameter to consider for evaluating the circuit's dynamic power dissipation, as most of the dynamic power dissipation is caused by the gate state transitions.

Switching activity can be defined as the ratio of the number of gate state transitions to the total number of clock cycles in the circuit.



$$P_{switch} = \frac{Vdd^2}{T} \sum_{n=1}^{num\_nodes} (\alpha_n * C_n)$$

Figure 1 - Activity Switching in CMOS

In Figure 1 is reported the formula for the switching power, where α_n represents the switching activity, C_n the load capacities, T the period of the clock and Vdd the power supply.

Switching activity can be reduced with clock gating techniques that switch off the registers clock and relative logic when not used.

Short-circuit currents are currents that flow through the circuit when two or more transistors conduct simultaneously, creating a direct connection between the power supply voltage and ground. This phenomenon increases with voltage and frequency.

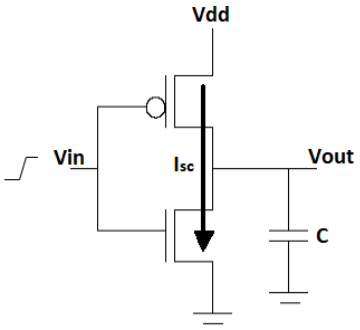
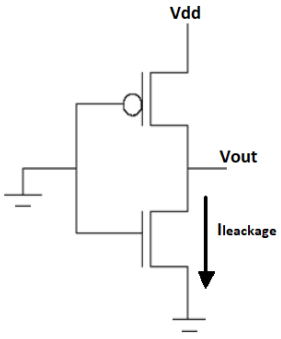


Figure 2 - Short-circuit current in CMOS

Ideally, CMOS gates draw zero current when off, but it is not true because the transistor leaks current when it is not activated.
The main leakage sources are:

- Subthreshold current between source and drain.
- Current through the thin gate oxide
- Current between drain/source and body inverse polarized junctions.

The main component of leakage current is the subthreshold current.



$$\text{Ileakage} = I_{D0}\frac{-V_{TO} + \eta V_{DS} - K_\gamma V_{SB}}{\eta V_{th}}$$

Figure 3 - Leakage current in CMOS

In Figure 3 the formula for subthreshold leakage current, where:

I_{D0} is current for VGs = VTh, η is the Drain-induced barrier lowering (DIBL) coefficient and K_γ is the body effect coefficient.

From the formula we can see that:

- Increasing threshold voltage decreases leakage.
- Decreasing VDD (voltage scaling) decreases leakage.
- Body bias polarization can help to reduce leakage.

A way to reduce the leakage is to use low-leakage cells if available.

Some technologies offer different flavor of cells: multiple-threshold cell library contains cells that are logically the same but have different threshold, so in the implementation phase the tool selects low-Vth cells on critical timing path while high-Vth cells can be used to reduce leakage.

1.2 Glitches power consumption

In an ideal world, the ports have zero propagation delay, so the switches are all perfectly controlled. In reality, ports sometimes make spurious transitions, called glitches, when inputs do not arrive simultaneously. Glitches cause additional power dissipation.

During glitching, voltage levels in the circuit can momentarily deviate from their expected values, causing the system to behave unexpectedly.

The power of glitches can be accurately estimated by means of a simulation that takes into account operating times.

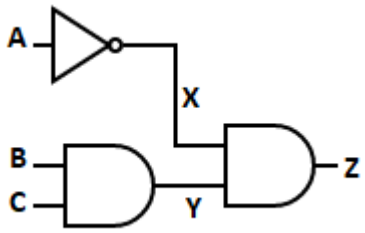


Figure 4 - Logic example for Glitches

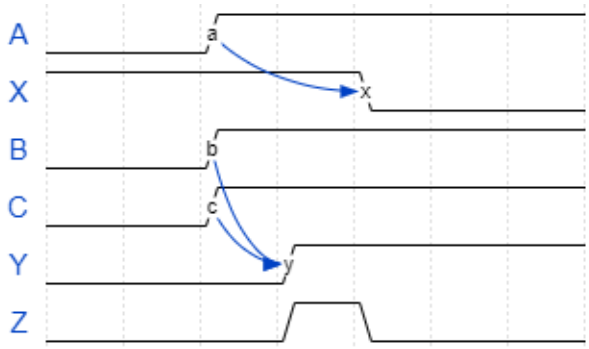


Figure 5 - Waveforms Glitches example

Figures 4 and 5 show a purely illustrative case of a possible glitch. The NOT port (on signal A) has a longer delay than the AND port: this condition generates for some time a logic high pulse on output Z.

In RTL design glitches can be reduced by breaking data path with registers to block propagation.

In the postlayout netlist a technique could be to equalize the timing path in order to reduce the unbalanced path.

2. Prelayout power estimation

For accurate power estimation we need to analyze the post layout netlist using GLS (Gate Level Simulation) switching activity.

Below is shown the flow.

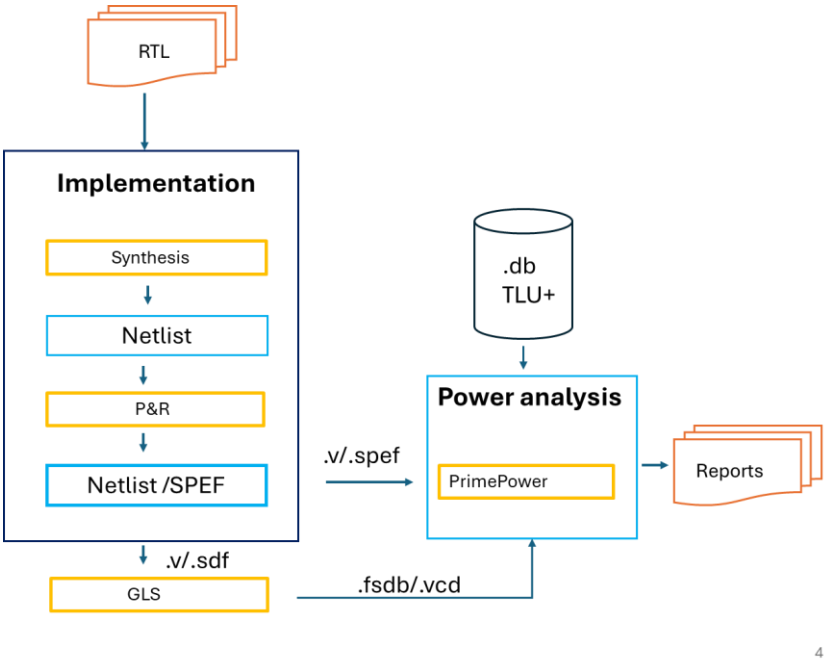


Figure 6 Postlayout power analysis.

Such flow requires execute the P&R and run GLS simulation.

If the power requirement is not met at this stage, it could be very hard to reduce the power with ECO.

On the other hand, an early-stage power estimation allows to anticipate and optimize power consumption at RTL level; the trade off is the accuracy.

At early stage of design some implementation information are not available for accurate estimation.

E.g.:

a) Clock tree is not inserted in the design: only synthesis netlist could be available.
b) GLS simulation environment isn't yet setup: only RTL switching activity is available.

One approach for early power estimation is to use synthesized netlist as input for PrimePower, however it lacks of accuracy due to missing clock tree and GLS switching activity: only RTL stimulus are available at this stage.

To estimate clock tree power in pre-cts netlist PrimePower provides the command

```
estimate_clock_network_power   <CT cell>
```

Based on the constraints and the type of the buffer specified (CT cell), RTL-PP inserts buffers for all the nets in the clock network and builds a virtual balanced clock network for power calculation.
The output load of each buffer is calculated using the wire load model, and the input transition of each buffer is propagated from the root.

However, results may be not reliable since information about cells placement is missing, and hence, a clock tree synthesis estimation can be totally different from postlayout one.

Moreover, estimated clock network feature is supported only in averaged power analysis that could be less accurate, in some cases, than time based analysis.

RTL activity from simulation can be annotated to post synthesis netlist using a map file generated during the synthesis step.
The mapping file can be extract from Formality RTL vs Netlist session through the command

```
write_register_mapping  -prime_power design_ppx.map
```

It contains the information on how RTL registers have been mapped on synthesized netlist:
i.e.

```
set_rtl_to_gate_name -rtl reg_hsel\[0\] -gate reg_hsel_regx0x/Q
```

Once the activity is mapped on invariant points (ports registers/memory outputs) the tool propagates it through the logic.

In order to well estimate glitch power, PrimePower allows to propagate the RTL activity taking into account the cell delay (information available in the liberty files), enabling the following variables

```
set power_analysis_mode time_based
set power_enable_delay_shifted_event_analysis true
```

This approach still lacks the effect of wire parasitics and clock tree skew, which are not negligible for switching power and in particular for glitch power.

3. Prelayout power analysis flow using RTL-A and RTL-PP

In the proposed flow the RTL-A and RTL-PP tools are embedded in the fronted kit used to generate the synthesized netlist (see Figure 7).

In such way, RTL-A inherits the same setting used for the synthesis, like timing, physical library, timing & power constraints, clock cells use lists, clock gating setting.

Nevertheless, it can exploit the same analyze/elaborate scripts used for synthesis to import the design.

Based on the previous two considerations we can see that the power estimation is highly correlated with postlayout signoff power data.

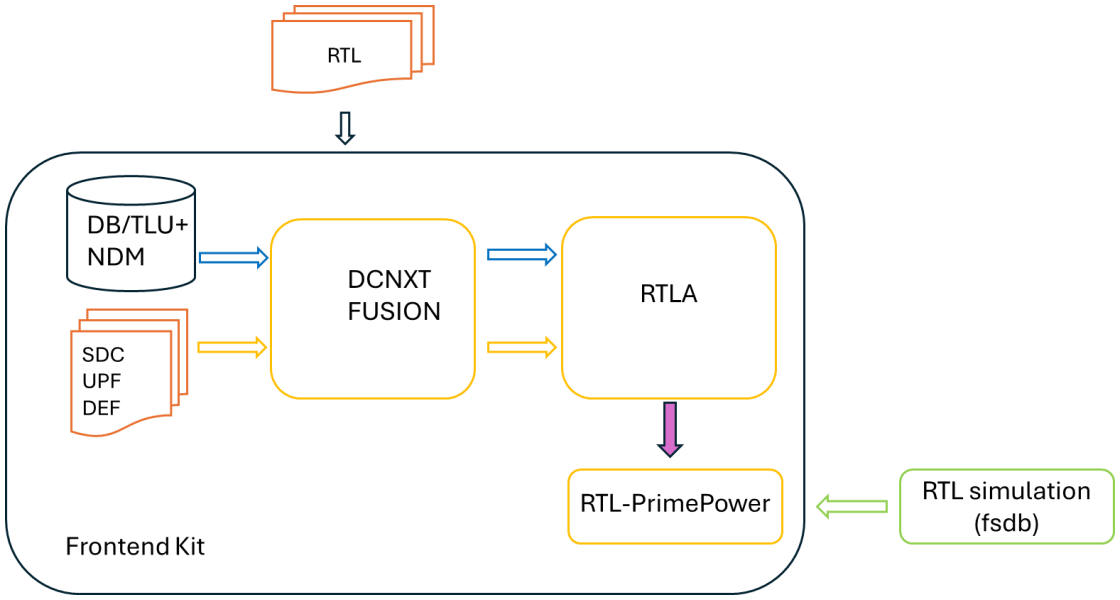


Figure 7 Fronted kit with RTLA- RTL_PP

Although it's possible to analyze power consumption with the RTL-A (report power metric), RTL-PP has been used for such analysis.
The advantages are:

- a more flexible flow that allow to use a synthesized netlist from a previous RTL Architect.
- run standalone PrimePower with the modified/multiple activity files (different operative modes)
- access to PrimePower capabilities not available or accessible in RTL Architect such as detailed Glitch Power Analysis, Self-Gating What-if Analysis, Clock Gating Efficiency Exploration

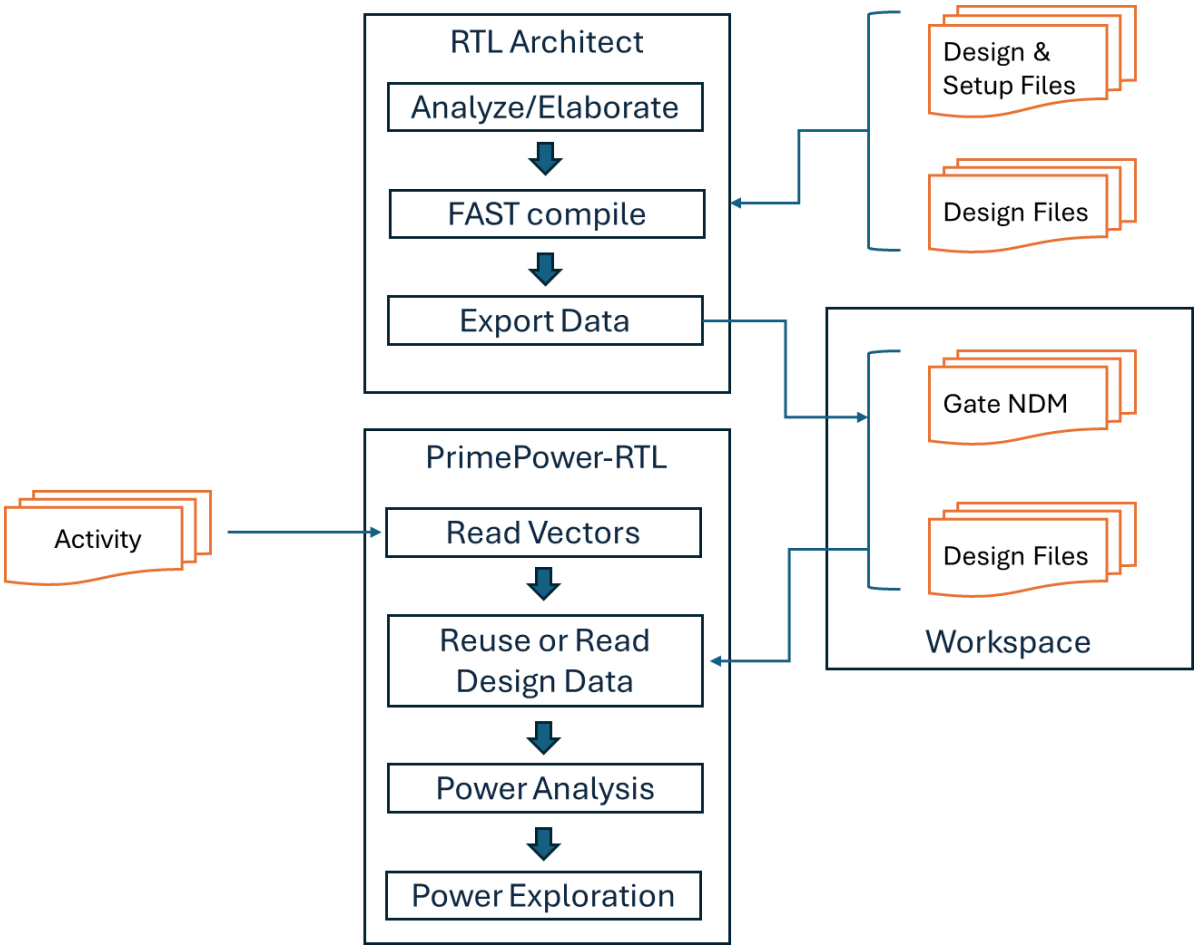


Figure 8 Hybrid RTL-A RTL-PP flow

Figure 8 shows the hybrid RTL-A RTL-PP flow:

Power analysis starts in the RTL-A tool and finishes in the RTL-PP.

So, the first step is to execute the RTL-A using the FC (Fusion Compiler) setting already available in the Fronted Kit.

The Frontend Kit is an environment used for the front-end design phase. It is made of scripts for each tasks (Synthesis, DFT, Linting checks, Timing checks, Power checks), that shares a common setting files like, technology files (timing libraries, physical views), design constraints (timing and power) and configuration options (CT cells, clock gating options, DFT strategy, etc.).

As shown in Figure 9 the RTL-A scripts use the same FC setting of the kit: DEF, scenarios, UPF, clock tree cells.

In particular enabling the fast CT synthesis (`rtl_opt.flow.enable_cts`) all the FC clock_opt setup are included : Non-default routing rules , max-transition, max-capacitance and max-fanout.

Once the fast synthesis is completed (`rtl_opt`), after `compute_metric` for power is executed, the power data base is exported in $RTLA_PRIMEPOWER_WORKSPACE folder.

```
source USER_SETUP/read_files.tcl # RTL files list
read_saif  ( optional)
source ./SETTINGS/FUSION/compile_setup.fc.tcl
save_block
load_upf & commit_upf
source ./SCRIPTS/FUSION/scenarios.load.fc.tcl
read_def  (optional)
source ./SETTINGS/FUSION/clock_opt_setup.tcl
set_app_options -name rtl_opt.flow.enable_cts -value true
rtl_opt
set_rtl_power_analysis_options -pwr_shell  $RTLA_PRIMEPOWER_SHELL_ROOT  ;
set_scenario_status -leakage_power true -dynamic_power true -active true \

$RTLA_POWER_ANALYSIS_SCENARIO
set_rtl_power_analysis_options   -scenario   $RTLA_POWER_ANALYSIS_SCENARIO \
                          -fsdb    $RTLA_SWITCHING_ACTIVITY_FILE_FSDB \
                          -strip_path $RTLA_SWITCHING_ACTIVITY_STRIP_PATH \
                          -time           $RTLA_SWITCHING_ACTIVITY_TIME_WINDOW
                          -output_dir $RTLA_PRIMEPOWER_WORKSPACE ;
compute_metrics -power
export_power_data ; # Data are written into $RTLA_PRIMEPOWER_WORKSPACE cache
```

Figure 9 RTL-A basic script.

Figure 10 shows the RTL Prime Power basic script.

Once loaded the RTL-A power exported database is possible to analyze the power data.

```
source ./SETTINGS/RTLA/pprtl_setup.tcl # Setup PrimePower-RTL
read_design_data $RTLA_PRIMEPOWER_WORKSPACE # Load design using hybrid commands
read_name_mapping
set_app_var power_analysis_mode   time_based

read_fsdb -rtl        $PPRTL_SWITCHING_ACTIVITY_FILE_FSDB \
                  -strip_path $PPRTL_SWITCHING_ACTIVITY_STRIP_PATH \
                  -time       $PPRTL_SWITCHING_ACTIVITY_TIME_WINDOW

update_power
update_metrics
report_rtl_metrics
```

Figure 10 RTL Prime power script.

To analyze glitch power, specified setting have been applied as shown in Figure 11.

```
# Enable delay shifted analysis
set_app_var power_enable_delay_shifted_event_analysis true

##Enable cycle based glitches
 set_app_var power_enable_clock_cycle_based_glitch     true

#Align to GLS setting
set_power_delay_shifted_event_analysis_options \
-glitch_pulse_r 70 -effort_level high -include_external_delay  \
-write_activity_file   delay_shifted.fsdb
```

Figure 11 Glitch power setup

First of the delay shift analysis (DSA) feature of PrimePower needs to be enabled.

Once the DSA is enabled, PrimePower is able to shift all the events imported from RTL simulation vectors (zero delay): the timing analysis engine is able to calculate the cell/net delay of RTLA design database.

Looking at Figure 12 we see that thanks to delay shift feature, the glitch power can be estimated with more accuracy: with DSA disabled the glitch power is underestimated.

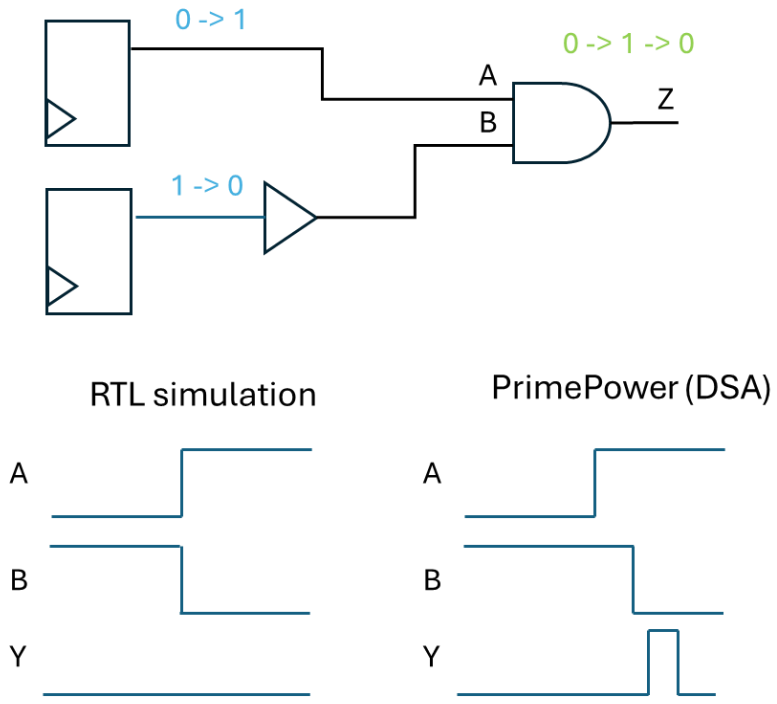


Figure 12 Glitches with shifted delay analysis

To improve glitches detection, we have enabled the clock-cycle-based glitch analysis, in this way all the pulses generated within a clock period are considered glitches.

To enable such option the variable `power_enable_clock_cycle_based_glitch` need to be set at true:
if this variable is disabled (default) these extra transitions are not labelled as glitches and are computed as generic dynamic power.
Enabling this variable helps to find the real glitch sources in the design.

Last setting in Figure 11 is related to filtering criteria in delay shifted analysis:
only the glitches whose pulse widths are larger than a certain percentage of cell delay, specified with `glitch_pulse_r` switch, are propagated to the next stage.
In the used script only glitches whose pulse widths are larger than 70% of the cell delay propagate throughout the design.

Note that to have the best alignment with postlayout power analysis, this pulse filter setting should be aligned to the one set in GLS simulation.

4. Power analysis result

The RTLA PP flow has been tested on a real design, which signoff power reports were available.

The design main features are:

Technology node: 130nm

Area: 180 Kgate

Max frequency: 32MHz

MACROs: 6xRAM , 2xROM

Below the cells type breakdown table.

Table 1 Cells type

Cell Type	Cell number	Cell %	Comment
combinational cells	**47148**	**81**	
sequential cells	**11216**	**19**	is the sum of below three cells type
gated registers	8807	15	84% of register
ungated registers	1672	3	18% of registers
clock gating	737	1	2 stage clock gating structure
Total core cells	**58364**	**100**	

Design has been implemented with DCNXT & P&R tool, while power signoff has been done with PrimePower.

Signoff power analysis has been performed using activity from backannotated GLS:

PrimePower has imported VCD file since time based power analysis was enabled.

Table 2 compares the power consumption by group of postlayout and RTL estimated in the worst power condition (fast corner, max voltage and max temperature) in operative mode.

Table 2 Postlayout vs RTLA power comparison by group

	postlayout	RTLA	
Power Group	Total [W]	Total [W]	Delta %
clock_network	4.59E-04	4.31E-04	-6
register	1.58E-04	1.38E-04	-13
combinational	4.74E-04	4.47E-04	-6
memory	6.19E-04	6.18E-04	0
Total	**1.71E-03**	**1.63E-03**	**-4.7**

We can see that the Delta is within 5%.

Thanks to the clock tree cell setting we can see a good match between the two clock tree power consumption.

We notice a perfect match on memory power consumption due to the same switching activity annotation between RTL and GLS simulation: RAM/ROM pinout is the same in RTL and postlayout netlist.

For the register group the difference is about 13%.

The reason of this mismatch is that the RTLA synthesis has used non-scan registers, while the postlayout netlist is DFT inserted and hence, most of the registers are of scan ready type: scan ready register is bigger and consumes more power than a register without scan pins.

A way to reduce such mismatch is to enable RTLA to use scan ready register.

Setting at TRUE the variable `compile.seqmap.scan` we see the delta reduced and register powers match (see Table 3).

Table 3 Postlayout vs RTLA (scan ready) power comparison by group

	postlayout	RTLA -scan ready	
Power Group	Total [W]	Total [W]	Delta %
clock_network	4.59E-04	4.24E-04	-8
register	1.58E-04	1.58E-04	0
combinational	4.74E-04	4.54E-04	-4
memory	6.19E-04	6.18E-04	0
Total	**1.71E-03**	**1.65E-03**	**-3.5**

Looking at the table below we can compare the glitch power.

Table 4 Postlayout vs RTLA power comparison by type

	Internal P. [W]	Switching P. [W]	Leakage P. [W]	Glitching P. [W]	Total P. [W]
Postlayout	9.71E-04	4.27E-04	3.17E-04	2.20E-04	1.71E-03
RTLA	9.65E-04	3.83E-04	2.86E-04	1.91E-04	1.63E-03
Delta %	**-1**	**-10**	**-10**	**-13**	**-4.7**

In the Table 5 the delta for the scan ready RTLA run.

Table 5 Postlayout vs RTLA (scan ready) power comparison by type

	Internal P. [W]	Switching P. [W]	Leakage P. [W]	Glitching P. [W]	Total P. [W]
Postlayout	9.71E-04	4.27E-04	3.17E-04	2.20E-04	1.71E-03
RTLA -scan ready	9.74E-04	3.79E-04	3.02E-04	1.89E-04	1.65E-03
Delta %	**0**	**-11**	**-5**	**-14**	**-3.5**

As we can see we glitching power (part of dynamic power) estimation is less about 14% than signoff value.

Using the RTL metric report is possible to rank the major glitch power contributors:

i.e.

```
>report_rtl_metrics -view rtl_file  -hier_attributes total_glitch -levels
15 -nworst 20  -sort_by  glitch_power_max


-------------------------------------------------------
RTL                      Glitch         Glitch
File                     Power          Power
Name                     Sum            Max
-------------------------------------------------------
rtl/dsp/mulhfp.v       7.517e-05      2.678e-05
rtl/dsp/addsub_hfp.v   2.075e-05      1.16e-05
```

Figure 13 report glitch power metric by file

The first two contributors are arithmetic blocks: in such logic the glitches are the main contributor to power consumption because of long data paths.

To further debug it is possible to cross probe the RTL source code in the RTL-PP GUI : as shown in Figure 14 the glitch power is reported next the RTL construct.

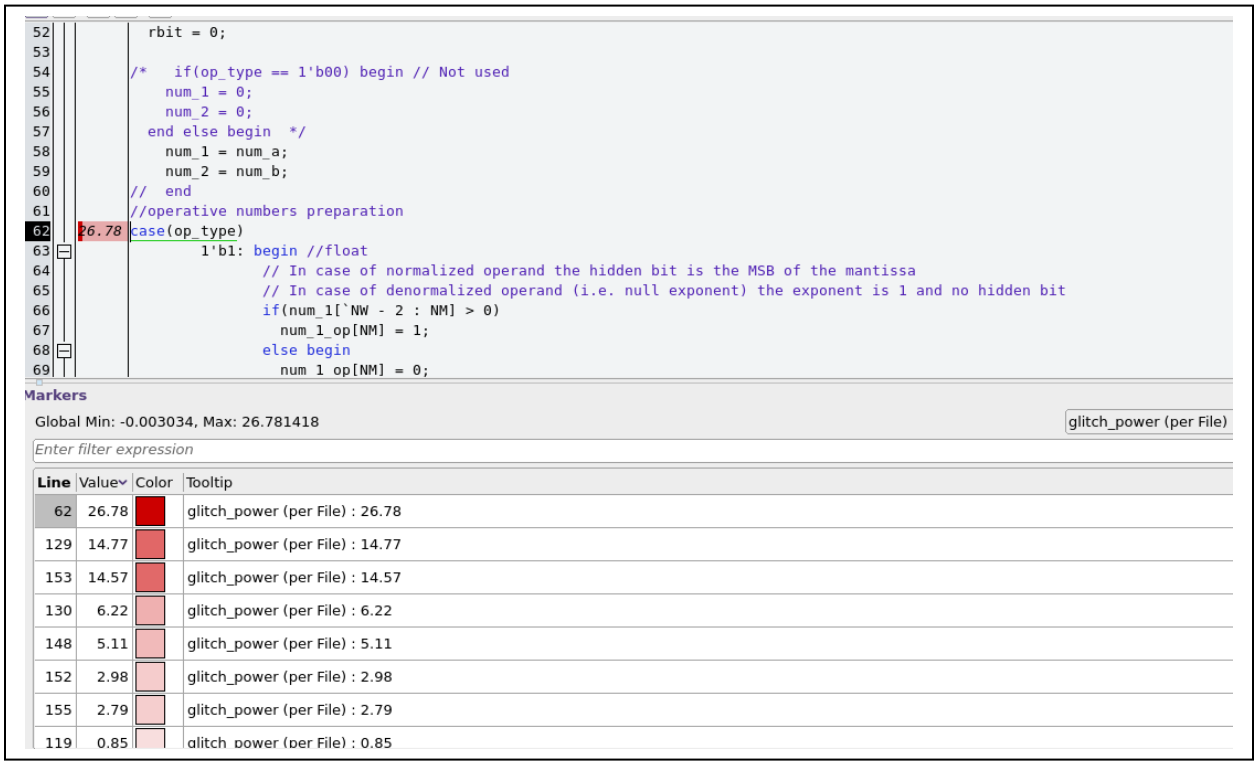


Figure 14 Glitch power RTL file cross probe

4.1 Glitch power optimization with RTL-A

Once identified the main glitch power sources, the possible optimization could be:

- Change the RTL design:
  adding register pipe on long data path to stop glitch propagation.
- Run a glitches aware synthesis:
  select low glitch designware architecture.
- Implement gate level optimization:
  fix unbalanced path, upsize/downsize cells.

Adding a register pipe in RTL code would have a strong impact on glitches reduction, the drawback is the increment of latency of one clock pulse for arithmetic operation, impacting the overall ALU performance.

Gate level optimizations are very dependent on netlist & layout, moreover there is the risk of worsening the timing: slow cells needed to suppress glitches can increase data path delay.

We have tried the second solution in order to keep the ALU performance and, at same time, to have a significative glitches power reduction.

Since the major glitch power contribute comes from a multiplier in DSP (Digital Signal Processor) we have replaced the default BOOT multiplier (best area /timing performance) with NAND based multiplier that is optimized for glitches (see [2]).

In Figure 15 it is shown the directive used for RTLA to choose a datapath multiplier architecture.

```
set_datapath_architecture_options  -mult_arch   nand \
                              [get_cells top/u_dsp/mul_module_inst]
```

Figure 15 Synthesis directives.

RTLA can use the Fusion Compiler directives for datapath optimization; for example, `set_datapath_gating_options` command could be used to gate di input of data path in order to block glitch propagation.

Table 6 shows the power comparison between the original RTL and the optimized one.
The glitch power has been reduced of 34%.

Table 6 RTLA power estimation on glitches optimized synthesis.

	Internal P. [W]	Switching P. [W]	Leakage P. [W]	Glitching P. [W]	Total P. [W]
RTLA	9.65E-04	3.83E-04	2.86E-04	1.91E-04	1.63E-03
RTLA-opt	9.34E-04	3.50E-04	2.85E-04	1.27E-04	1.57E-03
Delta %	**-3**	**-9**	**0**	**-34**	**-3.7**

In Table 7 the optimization run on RTLA scan ready.

Table 7 RTLA (scan ready) power estimation on glitches optimized synthesis

	Internal P. [W]	Switching P. [W]	Leakage P. [W]	Glitching P. [W]	Total P. [W]
RTLA (scan -ready)	9.74E-04	3.79E-04	3.02E-04	1.89E-04	1.65E-03
RTLA-opt (scan - ready)	9.37E-04	3.47E-04	3.01E-04	1.18E-04	1.58E-03
Delta %	**-4**	**-8**	**0**	**-38**	**-4.2**

5. Conclusions

The proposed frontend flow has proven to be a good alternative for early power estimation:

- Easy to setup: it is a plugin of existing front two back kit from which it inherits almost the settings.
- Accurate power analysis: within 5% with respect to postlayout signoff reports.
- Very fast flow: RTL -> RTLA run --> RTL-PP run X10 faster than a full signoff flow
- RTL metric reporting: detailed debug of many features/attributes allowing the cross probe to RTL code.
- It is possible to use the same synthesis optimization variables available in Fusion Compiler.

Further investigation can be done using the explore design feature of RTLA.

An idea could be vary the floorplan and let the tool evaluate the power consumption over all the provided DEF, comparing the results and choosing the best placement over PPA (Power, Performance , Area).

6. Acknowledgments

The authors would like to express their gratitude to STMicroelectronics for the opportunity to write this paper. They also acknowledge the very valuable insights provided by Claudio Mucci (STMicroelectronics) for the Fronted kit RTL-A development, and Alberto Baldi (Synopsys) for the support on RTL-A.

7. References

[1] CMOS VLSI Design 4th Edition: A Circuits And Systems Perspective By Neil H.E.Weste, David Money Harris. Publisher: Pearson

[2] Datapath Synthesis for Standard-Cell Design - Reto Zimmermann

[3] PrimePower and PrimePower RTL - User Guide

[4] RTL Architect™ User Guide

[5] Glitch Power Analysis with PrimePower