Tackling the challenge of simulating multi-rail macros in a power aware flow

Himanshu Bhatt	Amol Herlekar	Vikas Grover	Subhadip Nath
(Mgr., CAE)	(Sr. Staff, CAE)	(Sr. Mgr.)	(Sr. Design Engineer)
Synopsys, Inc.	Synopsys, Inc.	AMD, Inc.	AMD, Inc.
himanb@synopsys.com	herlekar@synopsys.com	vikas.grover@amd.com	subhadip.nath@amd.com

ABSTRACT

Digital and analog macros such as pll, serdes, and memory models have become an integral part of designs. For verification, macro vendors provide behavioral models that capture the macro functionality to some level of user-defined accuracy. For implementation, technology libraries (liberty syntax) are provided to the user. For functional simulation, behavioral models are integrated into the verification flow, while during implementation the macros get inserted for the corresponding behavioral models. This has been the traditional flow among design houses.

Low-power verification makes the problem more complex. The challenges are compounded if the macro is a multi-rail macro (having more than one power rail) or has an internal power switch, because in such cases, the macro cannot be partitioned into a single power domain. Assuming that the macro can be switched off, the objectives are to make sure that such macros can be accurately handled during low power simulation and that the simulator can expose issues like missing/incorrect connections, missing isolation cells or level shifters. The primary issue then is how to simulate macros in a power-aware flow such that the low-power information integrity is sustained.

Keywords

Low Power, Functional Verification, UPF
Unified Power Format, IEEE 1801

1. Introduction

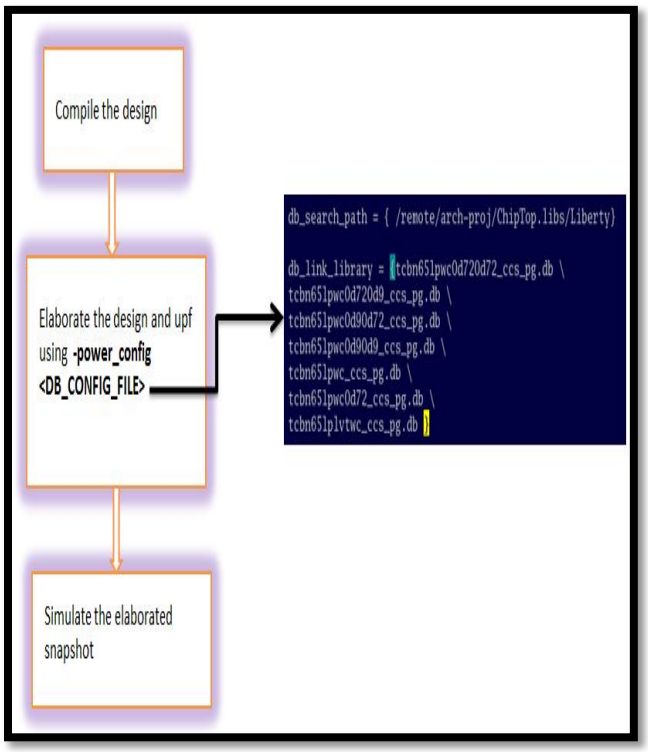
The solution to sustaining low-power information integrity when simulating macros in a power-aware flow requires a methodology that delivers accurate simulation results. The pace of design and the cost of silicon failure do not permit the electronics industry a similar turnaround time for power-aware design. The idea behind low-power verification is to catch any issues related to power-up and power-down sequences and to make sure that all required isolation cells, level-shifter cells, retention cells and corresponding policies in UPF are in place. All legal states and sequences must be covered and the design must never end up in any illegal state which it cannot exit.

This paper describes a methodology that makes catching bugs easier without actually imposing low-power semantics on the macro behavioral model. In this flow, macro .dbs (liberty files compiled and dumped by the synthesis tool) should be provided to the simulator. The simulator will use the .dbs to associate driving rails for each macro port because the .lib contains related power and ground pins for each logic port. The simulator will try to match the db instances with the macros in the device under test. If the PG ports are matched, then the behavioral model is said to be power-aware otherwise it is non-power-aware.

Once .db matching is done, the simulation semantics for such macros is that the UPF specification for the macro simply is ignored. No corruption semantics is applied to the internals of the model. No instrumentation for low-power semantics (e.g. isolation or retention) is done inside the model. This methodology thus helps avoid any undesired corruption or wake-up issues. Users can do a more accurate multi-rail macro simulation because ports are corrupted based on related power-down functions or related PG pins. Such issues as missing isolation cells will be caught in simulation because ‘x’ will propagate. This minimizes the risk of subtle bugs escaping into silicon.

This paper focusses on the low power simulation semantics of digital models only. AMS/analog models simulation is out of scope of this paper.

Simulation flow using macro with .db



This paper provides successful examples of this methodology in finding low-power functional bugs.

2. Low-power simulation of macros using .dbs

The macro models can be supplied in both Verilog behavioral codes and/or in implemented .dbs.
There are two kinds of these models:

- Power-aware needs to model accurately the effect of the supplies on the internal logic and also the relationship between input supplies and the outputs of the block.
- Non-power-aware is the standard behavioral model typically provided, with no supplies defined.

Power-aware model

This represents pre-implemented blocks with supplies. The corresponding RTL model has PG pins defined.

The PG pins in RTL can be declared as

- input/output ports
- UPF::supply_net_type
- reg/wire/logic
- supply0/supply1

The supply connections to the PG pins will be made

– Explicit – based on connect_supply_net in UPF
– Implicit – tied to primary supplies of the domain

The 'corruption' (or not) is handled by the model itself

Example: if (VDD1! == 1'b1) Q = 1'bz

Because the model is assumed to define its power-aware behavior, the simulator will not apply any corruption on these models.

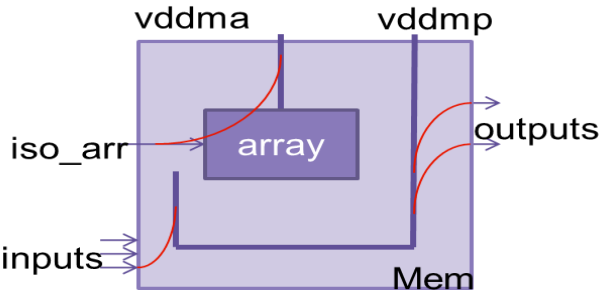
Non-Power Aware Model

Here the corresponding RTL model does not have the PG pins. Supply connections to the PG pins will be made

– Explicit – based on *connect_supply_net* in UPF
– Implicit – tied to primary supplies of the domain

The simulator will apply corruption on logic pins if power_down_function for the output pins evaluates to TRUE or in case of no power_down_function, if corresponding related_power_pin/related_ground_pin is turned OFF

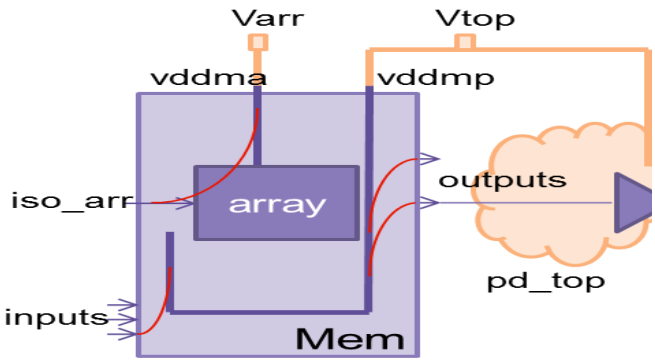
Example of a multi-rail memory



Behavior:

```
…
if shutoff (vddmp)
  corrupt_periph
  if (iso_arr == 0)
        corrupt_array
…
```

- Internal behavior depends on inputs and supplies (vddmp / vddma)
- Array can be retained if vddma is ON and iso_arr = 1



- Signals are connected in RTL
- Supplies are connected in UPF

connect_supply_net Vtop –port u_mem/vddmp
connect_supply_net Varr –port u_mem/vddma
set_domain_supply_net pd_top –primary_power Vtop

Example of a MRM Liberty Model

```
library(MACRO1_LIB){
 …
 cell(MACRO_MR){
  area : 37161.6;
  cell_leakage_power : 0.268402;
  pg_pin(VDDP) {
                voltage_name : VDD;
                pg_type : primary_power;
  }
  pg_pin(VDDA) {
                voltage_name : VDD;
                pg_type : primary_power;
  }
 …
  pin(A){
   capacitance : 0.02372;
   direction : input;
   related_power_pin : VDDP;
   related_ground_pin : GND;
}
  pin(SLEEP){
  ….
```

The above figure shows the definition of the macro cell. The supplies and related power and ground pins are defined.

We refer to this methodology as “simulating the macros for port-based corruption using .dbs.”

Macros with internally switchable models should preferably have complete PA models. Otherwise tool will do port corruption based on power-down function or related supply.

2.1 Methodology details

The methodology is to provide macro .dbs (liberty files compiled and dumped by the synthesis tool) as input to the simulator, which uses this information to simulate the models depending on whether these are power-aware or non-power-aware.

Information related to .dbs (search path and .db name) can be provided in a configuration file:

db_search_path = {path1 path2 path3}
db_link_library = {db1 db2 db3}
This .db configuration file can be passed to the simulator during compilation as follows:
%vcs –upf <UPF_FILE> <DESIGN_FILES>
-power_config <DB_CONFIG_FILE>

Sample .db_config file

```
db_search_path = { /remote/arch-proj/ChipTop.libs/Liberty}

db_link_library = {tcbn65lpwc0d720d72_ccs_pg.db \
tcbn65lpwc0d720d9_ccs_pg.db \
tcbn65lpwc0d90d72_ccs_pg.db \
tcbn65lpwc0d90d9_ccs_pg.db \
tcbn65lpwc_ccs_pg.db \
tcbn65lpwc0d72_ccs_pg.db \
tcbn65lplvtwc_ccs_pg.db }
```

The tool will try to match the .db instances with the macros in the device under test. If the following conditions are specified then a .db cell is said to have matched with the behavioral model:

- Name of the macro behavioral model and the .db cell must match.
- All macro behavioral model ports must match the .db cell logic pins in name and width.
- If there are PG ports in the model, they must match .db PG ports and should not have a width of more than 1.

If the PG ports are matched, then the behavioral model is said to be power-aware. Otherwise, it is non-power-aware.

Once .db matching is done, the simulation semantics for such macros are the following:

- Any UPF specification for the macro is ignored.
- For all such macros, no corruption semantics is applied to the internals of the model. This helps avoid any undesired corruption or wake-up issues.
- No instrumentation for low-power semantics (e.g. isolation or retention) is done inside the model.
- If the behavioral model is power-aware, it is assumed that the model internally takes care of corruption, and so the simulator does not instrument anything for corruption. Only the tool drives appropriate values on the PG ports of the model. This is the default behavior and can be overridden to instrument corruption by the simulator.

Example of a power-aware model:

```
module BUFFD0HVT (I, Z, VDD, VSS);
input I, VDD, VSS;
output Z;
assign Z = (VDD) ? I : 1'bx;
endmodule
```

- If the behavioral model is non-power-aware, macro logic pins are corrupted based on power-down function, if specified, otherwise related power ground pins are used for corruption. Again, this is the default behavior and can be overridden to skip corruption.

  Example of a non-power-aware model:

```
module BUFFD0HVT (I, Z);
input I;
output Z;
assign Z = I;
endmodule
```

As described, as per this methodology, by default

- Corruption will not happen on power aware models
- Corruption will happen for non-power aware models

This methodology provides users the flexibility to override the default behaviors using the following UPF commands:

- Apply corruption on all cells:
  set_design_attributes –attribute { SNPS_override_pbp_corruption TRUE}
- Do NOT apply corruption on all cells:
  set_design_attributes –attribute { SNPS_override_pbp_corruption FALSE}
- Apply corruption on an individual cell:
  set_simstate_behavior ENABLE –model {model_name}
- Do not apply corruption on an individual cell:
  set_simstate_behavior DISABLE –model {model_name}

The preceding methodology describes the process as implemented and used in the Synopsys low-power simulation (MVSIM-NLP), but the general concept can be applied to any simulator.

2.1.1 Alternative solutions considered

We considered two other approaches. The first one is to treat the macros as “always on.”

Treat the macros as “always on”

Because the behavioral models are not going to be synthesized and eventually will be replaced by macros, is it worth implementing low-power simulation semantics on such behavioral models? Of course, you might find design issues in the model, but they might not be real design bugs. Also, instrumenting low-power semantics on synthesizable code in such models might be overkill for the tool. Such models might not be able to handle shut-down corruption. It might be difficult to infer resets/clocks and flops. There might be unwanted shut-down issues due to corruption because there are lot of $tasks and similar

constructs used in such models. You might end up debugging undesired wake-up issues.
The user can treat behavioral models as “always on” by using the following UPF command:

set_design_attributes –models <MODULE_NAME> -attribute UPF_dont_touch TRUE

Once you have marked the behavioral models as UPF_dont_touch, the simulator will not do any instrumentation inside it. The internals as well as the outputs of such models will not be corrupted by the tool. If there is some ‘x’ on the inputs of the model, only that ‘x’ will propagate through the cone of logic. This will avoid all the unnecessary corruption and its effects on the macro.
If the behavioral model is power-aware, then the voltage values can be propagated from the UPF to the model. In any case, if the model is completely power-aware, it does not make sense for the simulation tool to instrument corruption semantics on the model. The power-aware model will take care of the internal as well as output corruption based on the voltage values.
There is also a flipside for such an approach. Consider the case when such a model is non-power-aware. What if one of the outputs of such a macro has a direct sink in a relatively more “on” domain? Because we are not going to corrupt the outputs of this macro, simulation will not be able to catch issues like missing isolation cells. The same might be true for level shifters.

This mode should not be preferred for macros having internally switchable domains. It should only be used if the macro has been thoroughly verified at block level earlier.

The second alternative methodology is to simulate the macros with UPF.

Simulate the macros with UPF

Some users like to simulate such macro behavioral models with UPF for the sake of confidence. If UPF is provided, the tool instruments the behavioral model for corruption, isolation, retention and other low-power simulation semantics. In such a case, users can catch issues like missing isolation cells because ‘x’ will propagate. On the flipside

- The design issues that simulation catches in behavioral model might not be real bugs.
- For retention, flops might be inferred incorrectly in unsynthesizable code.
- Resets/Clocks might not be inferred correctly leading to power-up issues.
- Models might not have ability to handle corruption, so you might end up debugging non- issues.
- Because any power rail can have only one power-net and one ground-net, it might not be possible to simulate multi-rail macros with this approach, unless the macro forms the boundary of some power domain, when set_related_supply_net/set_port_attributes can be specified.

2.2 Use cases

2.2.1 Use model description at AMD

- Non-power-aware BFM models for macros were instantiated in the design.
- Corresponding .db files were passed to the simulator.
- UPF with necessary isolation policies were passed to the simulator.
- CSNs were specified in the UPF to connect the supply rails of the macros.
- Necessary power information was present in the .db files, while the BFMs were non-power-aware by themselves.
- Default NLP behavior for PBC was utilized in power-sequencing tests, i.e., models identified as power-aware were not corrupted, while non-power-aware models were corrupted.

Macro information:

- Around 2,000 unique .db files were passed to the NLP tool with db_link_library variable.
- The NLP tool matched about 500 unique multi-rail macro .dbs. (Breakdown: 100 with two power rails, 400 with three power rails, and a

few having six power rails, excluding ground rails).
- NLP applied this methodology on about 720 such macro instances.

2.2.2 Benefits from adopting this flow

- Higher CPS and lower peak memory for power-aware simulations.
- Improved debug ability of power issues and decreased debug cycle time because we were not required to verify undesired corruption and wake-up issues inside macro BFMs.

2.2.3 Useful bugs found

- Bugs related to power-down and wake-up sequences were detected and fixed as usual.
- The bug detection improvement may not be quantifiable due to a change of methodology of power verification. However, SNR was improved and fewer false alarms were hit.

Above all, this flow adoption gave the verification team a better confidence of signing off the low-power verification.

3. CONCLUSIONS AND FURTHER DEVELOPMENTS

This closed-loop methodology ensures that no bugs in a macro used for low-power simulation escape into silicon.

We conclude the following based on this flow:

Pros of .db methodology:

- Avoid undesired corruption or wake-up issues.
- More accurate multi-rail macro simulation because ports are corrupted based on related power- down functions or related PG pins.
- Catch issues like missing isolation cells because ‘x’ will propagate.

Cons of .db methodology:

- If the models have PG pins but not corruption instrumentation, they will be treated as power-aware models. In such cases, no corruption will be done.
- Using .db flow is recommended based on the assumption that the macro vendor has done accurate low-power verification for the macro and that there are no holes. If that is not the case, then this flow is prone to missing bugs.
- Issue like power-on reset not getting asserted for memories inside the macro cannot be caught because there will be no corruption, and valid data will be read out on power-up even though no data has been written into the memory.

4. ACKNOWLEDGMENTS

The authors thank the entire low-power simulation team, without whose support this flow would not have been a success.

5. REFERENCES

IEEE Standard for Design and Verification of Low Power Integrated Circuits