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

Himanshu Bhatt/Amol Herlekar
Synopsys Inc.

Vikas Grover/Subhadip Nath
AMD, Inc.

March 4th, 2014
Agenda

• Should macro be power-aware?
• Available macro power-aware simulation methodologies
• Simulate macro as “always on”
• Simulate macro with UPF
• Simulate macro with .db
• Use cases
• Conclusion
Should macro be power-aware?

• Macros like PLL, SerDes, memory, etc., are replaced with behavioral models during functional verification.

• Behavioral models are never going to be synthesized. Is it really necessary to impose power-aware simulation semantics on them?

• Another school of thought believes in thorough low-power verification of macros for the sake of verification confidence.
Available macro power-aware simulation methodologies*

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Behavioral Model</th>
</tr>
</thead>
</table>
| Simulate macro as “always on”    | 1. Non-power-aware  
                                 | 2. Power-aware                |
| Simulate macro with UPF          | Non-power-aware                  |
| Simulate macro with .db          | 1. Non-power-aware  
                                 | 2. Power-aware                |
Simulate macro as “always on”

- Because macros are replaced with behavioral models during simulation, any design issue caught on these models during simulation might not be a real design issue.

- Instrumenting low-power semantics on non-synthesizable code might cause unwanted shut-down issues, and turning existing user-defined tasks and checkers into power-aware is troublesome.

- MVSIM-NLP implemented the following command to make behavioral models always-on

  `set_design_attributes --models <MODULE_NAME> -attribute UPF_dont_touch TRUE`
Simulate macro as “always on”
Simulate macro as “always on”

• Another variation of this “always on” simulation methodology is to make the macro’s behavioral model power-aware.

```verbatim
module macro_behav ( 
  input wire power_supply,
  ... 
  output wire power_aware 
);
  ...
  assign power_aware = power_supply ? orig_signal : 1’bx;
  ...
endmodule
```
Simulate macro with UPF

- Unified Power Format is devised for power-intent specification.

```plaintext
create_power_domain PD_Macro
create_supply_port VCC
create_supply_port VSS
create_supply_net VCC --domain PD_Macro
create_supply_net VSS --domain PD_Macro
create_supply_net VCC_SW --domain PD_Macro
connect_supply_net VCC --ports VCC
connect_supply_net VSS --ports VSS
create_power_switch VCC_PSW --domain PD_Macro
    -control_port {ctrl Sleep} --input_supply_port {vin VCC}
    -output_supply_port {vout VCC_SW}
    -on_state {on_state vin Ctrl}
set_domain_supply_net PD_Macro --primary_power_net VCC
    --primary_ground_net VSS
set_related_supply_net --power {VCC_SW} --object_list {DOX}
set_isolation_iso_from_macro --domain PD_Macro
    --isolation_power_net VCC --isolation_ground_net VSS
    --clamp_value 0 --applies_to_outputs
set_isolation_control_iso_from_macro --domain PD_Macro
    --isolation_signal {iso_en} --isolation_sense high --location self
set_retention_ret_macro --domain PD_Macro
    --retention_power_net VCC --retention_ground_net VSS
set_retention_control_ret_macro --domain PD_Macro
    --save_signal {save high} --restore_signal {restore b low}
add_port_state VCC --state {ON 1.08}
add_port_state VSS --state {VSS 0.0}
add_port_state VCC_PSW/w --state {ON 1.08} --state {OFF off}
create_pst MACRO_PST --supplies {VCC VCC_SW VSS}
add_pst_state normal --pstate MACRO_PST --state {ON ON VSS}
add_pst_state sleep --pstate MACRO_PST --state {ON OFF VSS}
```
Simulate macro with UPF

- UPF is more flexible.
Simulate macro with .db

- Instead of creating a UPF file for each macro, leveraging the macro’s existing technology library files can save effort if this methodology meets the requirements.
- MVSIM-NLP is able to read the macro’s .db files and associate the driving rail for each macro port.
Simulation flow using macro with .db

Compile the design

Elaborate the design and UPF using `-power_config <DB_CONFIG_FILE>`

Simulate the elaborated snapshot

```
db_search_path = { /remote/arch-proj/ChipTop.libs/Liberty}
db_link_library = [tcbn651pwc0d720d72_ccs_pg.db \ 
    tcbn651pwc0d720d9_ccs_pg.db \ 
    tcbn651pwc0d90d72_ccs_pg.db \ 
    tcbn651pwc0d90d9_ccs_pg.db \ 
    tcbn651pwc_ccs_pg.db \ 
    tcbn651pwc0d72_ccs_pg.db \ 
    tcbn651pwc0d72_ccs_pg.db ]
```
Simulate macro with .db

• To match a .db cell with a behavioral model, the following conditions must be met:
  – Name of the macro’s behavioral model and name of the .db cell match.
  – All macro behavioral model ports match the .db cell’s logic pins in name and width.
  – If there are PG ports in the macro’s behavioral model, they must match db’s PG ports and their width should be 1.

• If all conditions are met, MVSIM-NLP treats the macro’s behavioral model as power-aware.
Simulate macro with .db

- The following two models are both power-aware.
- The waveform shows MVSIM-NLP won’t do any corruption instrumentation.
Simulate macro with .db

• The following behavioral model doesn’t match all conditions; thus, it is treated as non-power-aware.

module BUFFD0HVT (I, Z);
input I;
output Z;
assign Z = I;
endmodule
Overriding default behavior

• 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}
  ```
Use model description at AMD

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

• Around 2,000 unique .db files passed to MVSIM-NLP with db_link_library variable
• NLP tool matched about 500 unique multi-rail macro .dbs. (Breakdown: 100 with 2 power rails, 400 with 3 power rails, and a few having 6 power rails, excluding ground rails)
• NLP applied this methodology on about 720 such macro instances

Benefits seen from adopting this flow

• Improved debug ability of power issues and decreased debug cycle time because undesired corruption and wake-up issues inside macro BFMIs were not required to be verified.
• Improved SNR and fewer false alarms hit.
Conclusion

Pros of .db methodology

- Helps avoid any 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.
- Issues like missing isolation cells will be caught in simulation 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.
THANK YOU

QUESTIONS?