How UPF 3.1 Reduces the Complexities of Reusing PA Macros

Madhusudhana Reddy Lebaka, Abraham Guizer, and Progyna Khondkar,
Senior Staff Engineer, Senior Application Engineer, Senior Product Engineer, Mentor A Siemens Business

What's UPF?

- UPF is the ultimate power reduction methodology for – design, verification and implementation today.
- UPF provides the concepts and the artifacts of
  - ✓ Power management architecture,
  - ✓ Power aware verification methodologies and
  - ✓ Low power implementation mechanism.
- However, there are different versions of UPF available today!
  - UPF 1.0 (UPF Accellera 2007)
  - UPF 2.0 (IEEE 1801-2009)
  - UPF 2.1 (IEEE 1801-2013)
  - UPF 3.0 (IEEE 1801-2015)
  - UPF 3.1 (IEEE 1801-2019)
- But the Question is Why?
  1. New release do not 'totally' obsoletes previous releases.
  2. Verification and implementation tools supports different variations of UPF for different reasons?

What are the Problems with Macros?

For **Soft Macros**

- UPF is mandatory to model the outside env view
- Implementation is hierarchical but verification requires full SoC level flat view.

This expose conflict for implementation Vs verification

For **Hard macros**

– While implementation - do not use UPF even though they are power aware and contains isolation, power switch, power states etc. internally.

– While delivering – they are usually HDL behavioral model accompanies with Liberty libraries.

Liberty defines only few power architecture/interface characteristics like related supply on logic/pg pins.

To overcome gaps between physical interpretations, potential conflict verification and implementation – we need to understand

What are minimum boundary parameters – mandatory to integrate, verify and reuse macros with the entire system level design?

In Practice

SM comes only with power management constraints

HM comes with pre-defined UPF and

Both are pre-verified at IP (block) level.

Hence – objectives with IPs for implementation and verification and reuse are

(1) Connecting the IP to proper supplies,

(2) Ensuring power boundary,

(3) Protecting the boundary with proper strategies and

(4) Finally verify power states or power cycles with the entire system level design becomes mandatory.

Soft Macros

Soft Macro are designed in synthesizable RTL and part of a larger RTL subtree before implementation

Self-contained UPF is mandatory for implementation

- This is to accurately model the outside environment of **SM** based on the internal power supplies.
- Because implementation is hierarchical but verification requires full SoC level flat view

UPF perspective, **SM** is RTL and associated with the UPF attribute **{UPF_is_soft_macro TRUE}**

UPF accompanied with a **SM** must be complete, define its own top level power domain with **create_power_domain -elements {.} is_atomic** commands

Hard Macros

Hard Macro are already synthesized, or placed&routed

HM are silicon proven and comes with pre-defined UPF and pre-verified at IP (block) level.

(a) .v behavioral model or GL netlist, lib and UPF with own top power domain defined **create_power_domain -elements {.}.**

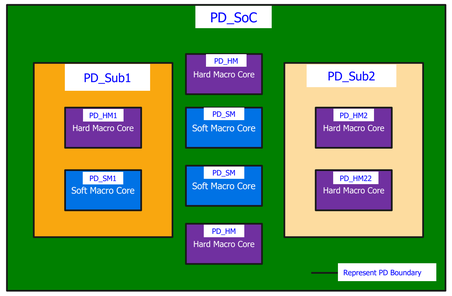
(b) .v behavioral model or GL netlist, .lib and UPF without own top level power domain

(c) .v behavioral model or GL netlist and .lib

UPF perspective, **HM** is IP block instantiated in the design with attribute **{UPF_is_hard_macro TRUE}** or **<is_macro_cell:true;>**

Only the supplies, IO pins and ports of **HM** are visible or accessible for integration and verification.

Hard & Soft Macros Integrated in SoC



How to Resolve Macro Problems I

- Know the boundary condition clearly:
  - Boundary **'create_power_domain PD -elements {.} -is_atomic' Vs begin /end /define /apply_power _model**,
- Terminal boundary – hard stop of everything!
  - Global supply net can't cross a terminal boundary
  - Parent context's power intent have no affect across a terminal boundary
  - UPF of an ancestor context can't contains any UPF artifacts of the child side of terminal boundary
  - e.g. power states, refinements of states, connection

How to Resolve Macro Problems II …cont

- All IO Ports treated as driver/receiver supply
  - Parent context (output driver/ input receiver supply)
  - Macro context (input driver / output receiver supply)
- When UPF is missing!
  - Anonymous power domain created around Macro with primary supply from parents
  - When Hard macro not specified in –elements{} list of any power domain
- Anonymously created domain boundary implies all terminal boundary conditions

How to Resolve Macro Problems III …cont

- Another important verification parameter is **corruption**
- **set_simstate_behavior <ENABLE | SIMSTATE_ONLY | PORT_CORR_ONLY | DISABLE>**
- Corruption semantics based on **set_simstate_behavior ENABLE** + Soft (or) + **Hard** Macro & **Liberty**
- Both - port & simstate corruption semantics applies
  - When SIMSTATE_ONLY + PORT_CORR_ONLY - **both enable** or
  - When SIMSTATE_ONLY enable **but** PORT_CORR_ONLY disable or
  - set_simstate_behavior DISABLE -models
- Strategy inside macros – Retention, Isolation –location fanout – **Applies** accordingly
- Hierarchical Macro domain – **Not allowed**
- For Anonymous domain - Tool implicit connects **HM** PG pins to parent domain primary – **domain primary corruption applies**

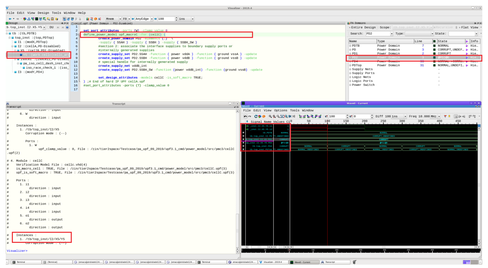
PA Soft Macro Verification

Power domain of **SM** created with atomicity signifies domain can be merged but cannot be split during implementation.

Contrarily, power model defined within **begin ~ end** or **define_power_model {}** and instantiated in a design with {**UPF_is_soft_macro TRUE**} may also represent a soft macro.

The rest – just follow the boundary condition

Even for **SM** the terminal boundary conditions apply



- SM instantiated in '**tb.top_inst.I2.X5.Y5**'
- Atomic power domain PD2 with **define_power _model**
- Reports/wave from simulation results
- Power domain status results

PA Hard Macro Verification

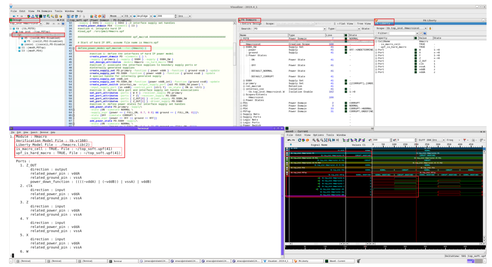
Power domain of **HM** created with hard boundary

Whether with create_power_domain or power model defined within **begin ~ end** or **define_power_model {}** and instantiated in a design with {**UPF_is_hard_macro TRUE**} may also represent a soft macro.

The rest – just follow the boundary condition

Particularly for **HM** the terminal boundary conditions makes it pure block box

Which makes How to Resolve Macro Problems III **difficult**



- **HM** named '**Hmacro**' instantiated in '**tb.top_inst.Hmacroinst**'
- Power domain with **define_power_model {}**
- Reports results (where simulation consider power model as **HM** cell)
- Power domain status, variables are also shown

Macro & Verification Env Reuse

Boundary conditions and constraints allows to reuse the verified hard macros across different projects.

Terminal boundary = power domain boundary for both **HM** & **SM**, plays significant roles PA verification.

- Developing dynamic custom verification env
- Populating UPF objects with **find_object** tracing and traversing through HDL for objects.

Remember, **find_object** for source or sink supplies for a particular signal will returns different results at 'core' and at 'SoC' level because of terminal boundary

Once these complexities are understood – reusing the macros and verification env become simple

Looking Back

- ✓ Standard SoC 'design cores' comes first and 'sub-system' integration follows these cores
- ✓ Depending on Spec and requirements, the 'cores' could be **Hard** or **Soft** macros.

Predominant Factors	Affecting DVIF
The extents of power domain boundary	Integration, Verification, Implementation
Terminal boundary	Implementation
Ancestor-descendant relations	Verification, Implementation
Power intent confinement	Verification, Implementation
Driver-receiver or related supply contexts	Integration, Verification, Implementation
Power states expectation	Integration, Verification
Simulation state behavior	Verification
Corruption semantics etc.	Verification
Flat Vs Hierarchical design	Integration, Verification, Implementation