

# Power Models and Terminal Boundary: Get your IP Ready for Low Power

Progyna Khondkar, William Winkeler, Phil Giangarra and Brandon Skaggs  
{progyna, billw, gphil @cadence.com}, Brandon.Skaggs@infineon.com

*Abstract-UPF power model plays vital role for soft and hard macro (i.e., IPs) verification and integration. On the other hand, terminal boundary typically signifies the power domain boundary for soft and hard macros and also plays significant roles in low power verification. However, understanding the self-containment of macros within power models, imply power intent object relations with ancestors, like supplies, strategies, source sink analogy, power states correlation and ultimately establish the terminal boundary concepts is utterly cumbersome to comprehend. This paper identifies and resolves the fundamental problems of IP design for low power systems and clearly shows with real examples, how the association of terminal boundary significantly simplifies to verify and integrate such IPs in larger systems.*

**Keywords**—UPF; Self-containment; Hard, Sof Macros; UPF power model; Treminal Boundary; UPF objects

## I. INTRODUCTION

The IEEE 1801 standard specifies the low-power intent, i.e., UPF for any design. The *UPF power model* in the standard language reference manual (LRM) defines the power intent of a model - more specifically intents of soft macros and as well *possibly* of hard macros. On the other hand, the boundary formed around these soft or hard macro is termed as terminal boundary - as it terminates the effects of all most all regular UPF commands and imply new verification (simulation) as well integration semantics on boundary ports. For example, power intent on ancestor descendant contexts, global supplies, isolation, level-shifter strategies etc. on a terminal boundary implicates a completely new semantics.

The intention behind such strict implication imposed by the UPF standard (IEEE 1801 2018, UPF 3.1) LRM is to simplify low power design, verification, implementation, and integration. Our motivation is to clarify such new semantics by accurately defining the characteristics of low power macros and identifying appropriate use models that establish the foundation for intuitive, portable, and standard low power verification in a bottom-up integration perspective.

In this paper we intended to establish the transparent relations of *power model* and terminal boundary and there after conducting empirical research to identify and *reinforce* a complete perception of soft and hard macro design, verification, implementation, and integration, as well *reuse* environment. With real design examples, we planned to exercise all the predominant factors that govern the simple and manageable macro verification and integration solutions. Evidently such effort will not only reveal best-practice methodology but also expose limitations in existing IP verification flows. In order to do so, we will first explain what soft and hard macros are and how they appear in real low power design verification, integration and implementation environment. And thereby we will show how self-contained *UPF power models* are used in conjunction to the complicated terminal boundary concept to make the entire UPF based design, verification, and integration straightforward for larger systems. Followings are the foundational discussion that will connect readers to the succeeding sections.

**UPF:** The Unified Power Format (UPF) also known as IEEE-1801, is not just a language to denote low-power intents or power specifications for a design – it’s a complete command set for verification and implementation of such low-power designs. The UPF is the ultimate abstraction of low-power methodologies today. It provides the concepts and the artifacts of power management architecture, power aware verification and low-power implementation for any design. It provides the notions of power architecture from very early stage of design abstraction. Now it’s the industry trend and standard for lowering static and in some special cases dynamic power dissipation in every digital design. Overwhelmingly UPF standout as the only alternative choice of lowering power dissipation when fabrication process technology advanced below 65nm.

**Hard Macro:** Hard macros are pre-implemented design blocks, essentially black box, available in behavioral RTL format without logic detail and associated with liberty library when instantiated in larger system level contexts for verification. For implementation, its available in netlist (LEF/GDSII) format, accompanied with liberty and mostly without UPF. So hard macro forms a terminal boundary which terminates all sorts of power management and

functional amendment to the block. Since this is already implemented and verified in smaller or self-contexts, a self-contained UPF may not be mandatory but usually ships with ‘*UPF\_is\_hard\_macro*’ attribute set to true. Such UPF specification comes without its own top-level domain but with driver/receiver supply for parents-context (outside), as well sometime for self-context (inside) ports. From UPF perspective, pre-synthesized netlist, liberty library and (or) minimum set of UPF is necessary for hard macro verification in bigger contexts. Obviously, there is no further implementation is required for hard macros.

**Soft Macro:** Soft macro are pre-synthesized or not yet implemented design blocks, which are available in synthesizable RTL and essentially part of a larger RTL subtree. Even though soft macros are not black box, they are not refinable – neither from UPF nor from RTL perspective. For example, remove or modify isolation location. So just like hard macro, a terminal boundary forms around soft macro which terminates all sorts of power management and functional amendment to the block. However, differ from a hard macro, a self-contained UPF that will define its own top-level power with ‘*create\_power\_domain -elements {.}*’ and ‘*UPF\_is\_soft\_macro*’ attribute set to true is mandatory for soft-macro. From UPF perspective, a set of constraint, configuration, and implementation UPF are combinedly necessary for soft macro verification. On top of that a liberty library will be required for implementation (to synthesize or hardened to a particular target technology).

**UPF power model:** *UPF power model* is a self-contained UPF, which is used to define or encapsulate the power intent of a model. The models are essentially HDL, behavioral RTL and (or) liberty cells for soft and (or) hard macros. The encapsulation may confine one or more such models and consists of UPF key command such as ‘*define\_power\_model*’ or ‘*begin\_power\_model*’ and ‘*end\_power\_model*’ in combinations with other regular UPF commands for power domains, isolation, retention, power switches etc. The ‘*apply\_power\_model*’ is another UPF key commands that binds *power models* to the design instances and connects the interface supply set handles and logic ports of a *power model*. The ‘*apply\_power\_model*’ command allows to map a supply set in the parent scope to a supply set in the *power model*. This is done using the *-supply\_map* option. LRM also allows mapping a parent supply set to the *power model*’s supply set handle. Since the parent supply set is mapped to the model’s supply set handle, only supply set handles can be used inside of the *power model*. The *UPF power model* is the concept of self-containment of power intent for soft and hard macros.

**Terminal Boundary:** The terminal boundary is a new UPF concept of boundary (specifically power domain boundary) conditions that are applied around soft and hard macros. All sorts of UPF including *find\_object*, global supply, location fanout for strategies, create object from parent contexts etc. and of course functional related amendment inside terminal boundary are strictly prohibited. It is also important to know that the driver and receiver supply contexts for soft & hard macro-IO ports reveal that they are regarded as the terminal points or terminal boundaries with respect to the ancestor power domain instances – this is common ground for all verification and implementation tools. Such boundary conditions obviously paved the way to confidently sign-off larger systems as well reuse them across multiple projects.

#### A. Motivation and Contribution of this Paper:

Our core objective is to accurately define the characteristics of low power macros and identifying appropriate use models that establish the foundation for intuitive, portable and standard low power verification in a bottom-up integration perspective. In this paper we conducted empirical research to identify and establish a complete perception of soft and hard macro verification, integration and reuse environment. With real design examples, we exercised all the predominant factors that govern the simple and manageable macro verification and integration solutions. We hope this paper, will serve as reference point to prepare IPs for low power verification, integration as well implementation.

#### B. Organization of this Paper:

This paper is organized in the following structure. Section I introduces the key concepts and relevant UPF topics in conjunction to the proposition and methodology. Section II explains fundamentals of UPF power model. Section III provides further insight of macros in terms of terminal boundary. The final sections IV draws the conclusion. The references are shown at the end.

## II UNDERSTANDING THE SELF-CONTAINMENT UPF FOR MACROS

As defined in section I, *UPF power model* are the container for self-contained UPF that can be defined with ‘*define\_power\_model*’ or ‘*begin\_power\_model*’ & ‘*end\_power\_model*’ duo (the later duo became legacy in latest 1801-2018, UPF 3.1 LRM) in conjunction with ‘*apply\_power\_model*’ command. Even though semantically the

current or the legacy commands carries same concepts, the syntax are such that, they encapsulate other regular UPF commands, like, power domains, supply sets, isolation, retention, power switches, power state, etc. within the curly braces for

*'define\_power\_model <model\_name> -for <model\_lists> {< UPF detail for the macro model>}'*  
and withing begin and end commands (now legacy in current LRM)

```
'begin_power_model <model_name> -for <model_lists>  
< UPF detail for the macro model >  
end_power_model'
```

For simplicity we will only discuss *'define\_power\_model'* but the explanation will be equally applicable for *begin/end\_power\_model* duo. The other important point to remember that the *<model\_name>* is the user provided simple name of the macro and *<model list>* actually lists the modules or liberty cells name of the macro. Following table shows example of a *UPF power model*, how its integrated in a parent or larger contexts in terms of UPF, and physical appearance of model or macro in a larger verification environment.

**Table 1 Understanding the UPF power model for Macros**

<b>UPF power model macro.upf</b>
<pre># Power model for macro define_power_model my_macro -for {cellA} {#start of encapsulation # Attribute to identify macro set_design_attributes -models cellA -is_hard_macro TRUE # Macros own top level create_power_domain PD_macro -elements {..} -supply {primary} # Associate interface supplies to boundary supply ports create_supply_set PD_macro.primary -function { power vdd } -function { ground vss } -update # Define power states for interface supply set handles add_power_state PD_macro.primary -supply \ -state {ON -supply_expr {(power == {FULL_ON,5.0}) &amp;&amp; (ground == {FULL_ON,0.0})} -simstate NORMAL}\ -state {NON -supply_expr {(power == {OFF}) &amp;&amp; (ground == {OFF,0.0})} -simstate CORRUPT} # Define Internal ISO, RET, Power Switch strategies ... } #start of encapsulation</pre>
<b>Top Level UPF top.upf</b>
<pre># UPF for design/dut interface that instantiate the macro # Design/DUT Power Domain create_power_domain PD_TOP -elements {..} ... # Supply set for Design/DUT Power Domain create_supply_set PD_top_ss -function {power VDD_TOP_net} -function {ground GND_TOP_net} create_power_domain PD_TOP -update -supply {primary PD_top_ss} # Instantiating the macro to design load_upf "macro.upf" apply_power_model my_macro -elements {dut/I1} -supply_map { { PD_macro.primary PD_TOP.primary } }</pre>
<b>Top Level design/dut, macro and instantiation of macro</b>
<pre>module wrap_tb(); top dut (clk,reset,inp,outh,buff); endmodule  module top (clk,reset,inp,outh,buff); .. supply_net_type vdd,vss; cellA I1 (vdd,vss,clk,reset,inp,temp); endmodule  module cellA (vdd,vss,clk,reset,inp,outh); &lt;other logics&gt; endmodule</pre>

Hence its' clear what the LRM denotes that a *"UPF power model"* can be used to represent one of the following:

A **hard macro**, indicated by the fact that the *power model* defines the attribute ‘*UPF\_is\_hard\_macro TRUE*’ on the model to which it applies. In this case, the UPF commands within a *power model* describe power intent that has already been implemented within the instances to which this *power model* is applied.

A **soft macro**, indicated by the fact that the instance to which this *power model* is applied has the attribute *UPF\_is\_soft\_macro TRUE*. In this case, the UPF commands within the *power model* describe power intent that remains to be implemented.

That’s the reason we mentioned in section I that for hard macro “a self-contained UPF may not be mandatory but usually ships with ‘*UPF\_is\_hard\_macro*’ attribute set to true.” So, for hard macro *define\_power\_model* is not mandatory, and verification tool can get power intent info or validate it from liberty or combinedly both liberty and *power model*. For Soft macro, *UPF power model* is mandatory. The requirement can be readily verified according to the UPF 3.1 as follows.

A **hard macro** may have a Liberty description, as well as one of the following:

- No UPF specification.
- A self-contained UPF specification.
- A UPF specification that does not define its own top-level domain. In this case, it shall be an error if any of its top-level ports’ driver\_supply or receiver\_supply is needed (by other commands in the UPF) but is not specified.

For **soft macro** it shall be an error if the UPF specified for a soft macro is not self-contained.

Before we move further to ‘*what/why is self-contained UPF*’, let’s understand the implication of ‘*apply\_power\_model*’ on hard & soft macro. This is equally important for macro verification as well integration on to larger system. A *power model* can be applied to specific instances using *apply\_power\_model*. According to LRM, this binds *power models* to the design instances and connects the interface supply set handles and logic ports of a previously loaded *power model*. This is exactly shown in Table 1 as follows.

```
# Instantiating the macro to design
load_upf "macro.upf"
apply_power_model my_macro -elements {dut/I1} \
                                -supply_map { { PD_macro.primary PD_TOP.primary } }

Syntax of apply_power_model is as follows.
apply_power_model power_model_name
[-elements instance_name_list]
[-supply_map {{instance_scope_supply_set current_scope_supply_set}*}]
[-port_map {{instance_scope_logic_port current_scope_logic_net}*}]
```

Where, the *apply\_power\_model* argument *model\_name* identifies the user specified simple name of a previously defined *power model*, e.g. *apply\_power\_model my\_macro*. The *-elements* option specifies a list of instances, relative to the current scope, to which the *power model* applies. If this option is missing, the scope is defined using the associated *define\_power\_model* or *begin\_power\_model* command. Specifically:

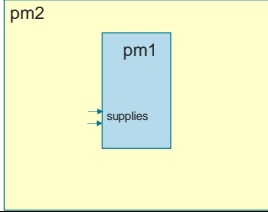
- If *-for* is used with *define\_power\_model*, then this option sets the scope.
- If a specified *-elements* instance does not match up with any of the model names in the *-for* list, verification tool must generate error.
- If *-for* is not used, then the *model\_name* sets the scope.

The *-supply\_map* option specifies how the supply set(s) from the instance scope in the defined *power model* associate with the supply set(s) of the current scope. The LRM specifies that, each *instance\_scope\_supply* and *current\_scope\_supply* pair implies an *associate\_supply\_set* command of the following format:

```
associate_supply_set {instance_scope_supply_set current_scope_supply_set}
```

The *-port\_map* option specifies how the logic/supply ports of the defined *power model* connect to the logic/supply nets in the current scope. Interestingly, UPF 3.1 LRM allows nested *power model* i.e. *one power model applied to a given instance may apply another power model to a descendant instance*. For example, applying ‘*apply\_power\_model*’ inside another *power model* can be done as follows.

**Table 2 Nested UPF power model for Macros**

Example of nested power model:	
<pre> define_power_model pm1 ... { .... } define_power_model pm2 ... { ....     apply_power_model pm1 ... ... } </pre>	

This eventually will also require multiple *load\_upf -scope* inside the *power model* for the nested *apply\_power\_models*. Note that a *power model* that is not referenced by an *apply\_power\_model* command does not have any impact on the power intent of the design.

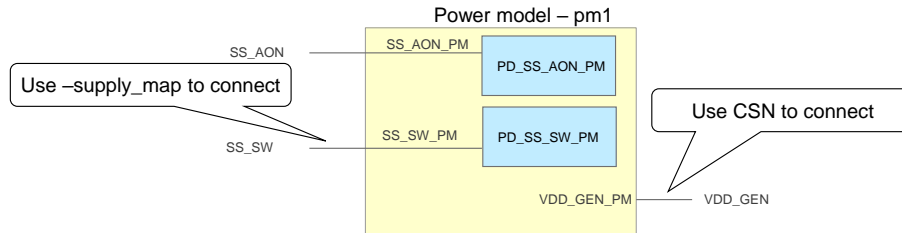
#### A. Why Self-Containment?

A larger system, for example, a SoC, consists of several macros and other synthesizable (or implementable logic blocks). While implementing such larger system it may be required to implement an instance separately from the top-level scope with the intention to integrate this block back into the system later in the flow. This flow style is often referred to as a bottom-up flow. If using a bottom-up flow, some considerations regarding UPF partitioning must be made. In particular the implementation of a lower-level instance will be done without the parent scope being present. Therefore, the block UPF power intent must be self-contained; in that it cannot rely on power intent defined in an ancestor scope and it cannot define power intent that is to be implemented in an ancestor scope.

A system may contain instances that have already been implemented or instances that will implemented separately. These instances in a bottom-up flow need to be defined as macros. By defining an instance to be macro, the evaluation of certain UPF power intent commands are affected since a macro forms a terminal boundary. In order for a block to be implemented standalone from its parent scope, the UPF for this block must completely define its own power intent. Hence self-containment is essential.

#### B. Case Study: Macro Integration for verification

In section I and II, we explained that the '*apply power model*' command allows to map a supply set in the parent scope to a supply set in the *power model*. This is done using the *-supply\_map* option. LRM also allows mapping a parent supply set to the *power model*'s supply set handle. Since the parent supply set is mapped to the model's supply set handle, only supply set handles can be used inside of the *power model*. This is explained in Figure 1 below.



**Figure 1 Macro integration from a larger system contexts**

**Table 3 UPF power model Connections to Parent Contexts (Supply Association)**

UPF power model and Parent supply association
<pre> # Power model and relevant UPF commands for inside the macro begin_power_model pm1 {     &lt;all upf commands that apply inside of the power model&gt; }  # Power model and Parent associated with Supply set handle apply_power_model pm1 -elements {list_of_instances} \     -supply_map {{PD_SS_AON_PM.primary SS_AON} \                 {PD_SS_SW_PM.primary SS_SW} ...}  # Power model and Parent associated with supply set directly apply_power_model pm1 -elements {list_of_instances} \     -supply_map {{SS_AON_PM SS_AON} \                 {SS_SW_PM SS_SW} ...} </pre>

The LRM also makes logic/supply ports of the *power model* connections with logic/supply nets connections in the parent contexts. Each pair in the *-port\_map* option implies either a *connect\_logic\_net* command or a *connect\_supply\_net* command depending on whether it's a logic connection or a supply connection. The connection format are shown below:

```
connect_logic_net current_scope_logic_supply_net -ports {instance_scope_logic_supply_port}
OR
connect_supply_net current_scope_logic_supply_netb -ports {instance_scope_logic_supply_port}
```

**Table 4 UPF power model Connections to Parent Contexts (Supply port map)**

UPF power model and Logic/Supply nets	
<pre># Macro Power Model define_power_model test_macro -for ip_macro { set_design_attributes -elements {..} -is_hard_macro true # Create Supply Sets create_supply_net VDD create_supply_net DVDD create_supply_net DVDDL0 create_supply_net VSS create_supply_net DVSS create_supply_set SS_VDD -function { power VDD } -function { ground VSS } create_supply_set SS_DVDD -function { power DVDD } - function { ground DVSS } create_supply_set SS_DVDDL0 -function { power DVDDL0 } -function { ground DVSS } # Define Power Domains create_power_domain pd_test_macro -elements {..}\ -s supply {first SS_VDD} \ -s supply {second SS_DVDD} \ -s supply {third SS_DVDDL0} associate_supply_set SS_VDD -handle pd_test_macro.primary</pre>	<pre># Parent Contexts create_supply_set SWCoreSupply \ -function [list power VDD_SW_top] \ -function [list ground VSS_SW_top] # Parents Domain create_power_domain PD_Parents -elements {..} \ -s supply {first SWCoreSupply } \ -s supply {second SWIOSupply } \ -s supply {third SWIoIntermediate } associate_supply_set SWCoreSupply -handle PD_IO.primary # Macro connection load_upf ./ip_macro.upf apply_power_model test_macro \ -elements {u1} \ -port_map { {VDD VDD_SW_top} \ {VSS VSS_SW_top} \ {DVDD DVDD_SW_top} \ {DVSS DVSS_SW_top} \ {DVDDL0 DVDDL0_SW_top} } }</pre>

Hence it's evident that the supply and logic connections are essential in order to hookup the macro through *UPF power model* for verification, integration and eventually implementation on to larger SoC contexts. The *apply\_power\_model* command describes the connections of the interface supply set handles of a previously loaded power model with the supply sets in the scope where the corresponding macro cells are instantiated. The arguments of the *-supply\_map* option need to be such that the implied *associate\_supply\_set* commands are legal. In addition, the *-port\_map* shows how the interface logic or supply ports of the instance scope (i.e. macro in *define\_power\_model* is defined) connect with logic or supply nets in the current scope (i.e. parent where *apply\_power\_model* is used) respectively. The arguments of the *-port\_map* option need to be such that the implied *connect\_logic\_net* or *connect\_supply\_net* commands are legal.

### C. Case Study: Macros from Successive Refinement Perspective

UPF supports the successive refinement methodology where power intent or UPF information grows along the design flow to provide needed information for each design phases. Hard macros are already implemented and hence not actually relevant for this methodology, although its relevant for verification. However, in bottom-up integration, successive refinement can play significant role for soft macro verification and implementation. In such flow, soft macro will require constraint, configuration and implementation UPF all together for verification and in addition, liberty will be required for implementation (i.e. synthesis/hardening after verification at RTL). In UPF perspective of IP verification, integration and implementation - such constraint, configuration and implementation UPF will remain within *UPF power model* through *define\_power\_model* commands.

We already discussed that soft macros follows self-contained UPF semantics and act as terminal boundary. Hence successive refinement flow is ideal for bottom-up integration and late implementation flows. However, successive refinement flow is not full-proof and there are known limitations, as noted in section I. Like soft macros are not refinable, for example, optimize or remove redundant strategies or alter locations of strategies inside. For example, remove or modify isolation location. Even the implementation UPF may not be altered for a different technology library mapping. The succeeding section will cover such limitations of in verification, integration and implementation

#### D. Limitations of Power model

In addition to the limitations noted above, there are other consequences of verifying soft macros at block or system level configuration. For example due to terminal boundary restrictions, user may not have access and control of the power enable signals within a terminal boundary defined in *UPF power model*. Even though UPF provides mechanism to create and connect logic nets and ports in a scope through *create\_logic\_net/port* and *connect\_logic\_net\_port* but such provisions are restricted. For example, consider Figure 2, where user may want to access different power control signal within IP1 terminal boundary as input ports only and as well associate a driver supply with the logic port to confirm that the correct supply is ON for the control enable signals.

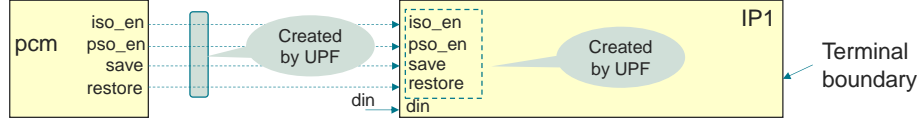


Figure 2 Macro verification from a larger system contexts

Table 5 *UPF power model limitation example*

#### User expectations in *UPF power model*

```
# UPF Extension for IP1 macro in Power model
create_logic_port iso_en
create_logic_port pso_en
create_logic_port save
create_logic_port restore
# associate a driver supply with the logic port
set_port_attributes -ports {iso_en pso_en save restore} -driver_supply SS1
```

In ideal situation, the *set\_port\_attribute* command for logic port and HDL port should have no functional difference. However, LRM restricts the extension of such input power enable ports with soft-macro and associate them with relevant driver supplies. One important aspect to remember that *find\_object* will still not work for accessing such logic ports unless *-traverse\_macro* is specified.

### III THE BOUNDARY CONDITION FOR VERIFICATION

#### A. Why Boundary Condition?

As explained in Section II, bottom-up integration and implementation are the only productive choice for SoC design today. Hence it became obvious to make some consideration regarding UPF portioning. In particular, the implementation of a lower level instance will be done without the parent scope being present. Therefore, the block UPF power intent must be self-contained. That is, it cannot rely on power intent defined in an ancestor scope and it cannot define power intent that is to be implemented in an ancestor scope. Since the power model must be self-contained as every aspect of larger SoC contexts are not available, the boundary conditions are specified for the model.

Now it's clear that whether design verification engineers are dealing with hard or soft macros, the parent context of the UPF (where *UPF power model* of macros are loaded and *apply\_power\_model* is used) specifies only those conditions as seen from outside of these macros. This is why *driver/receiver\_supply* are mandatory for soft and hard macros to specify appropriate *driver\_supply* for output and *receiver\_supply* for input ports from the outside of these macros (parent contexts). And exact opposite – i.e. *driver\_supply* for input and *receiver\_supply* for output from inside of these macros (self-contexts). However, inside declaration are optional for hard macros. But its obligatory to verify when or if they are present in accompanied *UPF power model*.

#### B. What are the Boundary Condition for IPs Verification, Integration and Implementation?

The boundary conditions for IPs (soft & hard macro) verification are already explained in previous sections combined with integration as well implementation perspective. Here in this section, our objective is to summarize these boundary conditions in standout formats so that it become very straightforward to prepare any IP (soft & hard) for low power verification, integration and implementation on to larger SoC contexts. Refer to Table 6 below that lists all these conditions.

**Table 6 Boundary Condition for IPs (Soft & Hard macros)**

Boundary Conditions for Ips
Self-contained UPF with own top-level power domain without any reference to any external objects
No UPF objects inside macro from parent contexts
No reference of power states
No reference of child scope objects inside macro
No visibility of real drivers and receivers for IOs from parent contexts
<i>driver_supply</i> for output and <i>receiver_supply</i> for input from parent-contexts
<i>driver_supply</i> for input and <i>receiver_supply</i> for output from self-contexts
location fanout stops at boundary
Isolation, Level-shifter location parent allowed
Isolation input cannot specify -sink Isolation output cannot specify -source
Level-shifter cannot specify input_supply, output_supply for self, other, fanout
No global supply reference from inside macro
find_object must accompany -traverse_macro
No modification in connect_supply/logic_net and -reconnect

### C. Generic Factors affecting boundary conditions

Even though we have explained the UPF perspective of integration and verification details of both soft & hard macros in previous sections, we realized that the methodological aspects that governs the macro integration and verification requires a standalone section for proper attention. In the previous sections, we have clarified the views, contents and contexts of soft and hard macros for power management perspective from system integration and verification perspective. In this section we will further clarify the additional power management semantics that are crucial for the entire design, verification, implementation and integration flow. Table 7 summarize the complete lists of all the predominant factors that affects and governs macro integration and verification.

**Table 7 Factors for Macro Verification, Integration and Implementation**

Predominant Factors	Affecting Verification, Integration and Implementation
UPF power model	Integration, Verification, Implementation
Terminal boundary	Verification, Implementation
Ancestor-descendant relations	Verification, Implementation
Driver-receiver or related supply contexts	Integration, Verification, Implementation
Power states expectations	Integration, Verification
Simulation state behavior	Verification
Corruption semantics etc.	Verification
Flat Vs Hierarchical design (reason of having terminal point)	Integration, Verification, Implementation

## IV CONCLUDING REMARKS

In this paper we attempted to establish a very transparent relations between *UPF power model* and terminal boundary and there after identifying and *reinforcing* a complete perception of soft and hard macro design, verification, implementation, and integration environment. With real design examples, we exercised all the predominant factors that govern the simple and manageable macro verification and integration solutions. We intended to reveal best-practice methodology but at the same time, also point out exact limitations in existing IP verification flows from UPF perspective. We intentionally avoided methodological perspective of soft and hard macro corruption semantics, as it requires an extensive research on liberty, *UPF power model*, *set\_simstate\_behavior* <ENABLE, SIMSTATE\_ONLY, PORT\_CORR\_ONLY>, inherited or parent power domain primary etc. based corruption perceptions and we consider pursuing such endeavor in a future research and publications.

## ACKNOWLEDGMENT

The authors would like to express sincere gratitude to the reviewers from Infineon Technologies and Cadence Design Systems for providing useful insight and guidance in conducting the research and preparing this paper.

## REFERENCES

- [1] Progyna Khondkar, "Low-Power Design and Power-Aware Verification", Hard Cover ISBN: 978-3-319-66618-1, October, 2017, Springer International Publishing.
- [2] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801™-2018.
- [3] Progyna Khondkar, et al., "How UPF 3.1 Reduces the Complexities of Reusing Power Aware Macros" March, DVCon 2020."
- [4] Progyna Khondkar, et al., "Low Power Coverage: The Missing Piece in Dynamic Simulation", February March, DVCon 2018.