How UPF 3.1 Reduces the Complexities of Reusing Power Aware Macros

Madhusudhana Reddy Lebaka, Abraham Guizer, and Progyna Khondkar,
Senior Staff Engineer, Senior Application Engineer, Senior Product Engineer, Mentor A Siemens Business
(madhul427@gmail.com, abraham_guizar progyna_khondkar@mentor.com)

***Abstract*-Integration of soft & hard macros with low power designs and conduct power aware (PA) verification are always complex and cumbersome. Specifically, in bottom up integration perspective, the extents of power domain boundary, terminal boundary, ancestor-descendant relations, power intent confinement, driver-receiver supply contexts, power states expectations, simulation state behavior, corruption semantics etc. for these macros were not well defined until UPF 3.1 (IEEE 1801-2019). As a consequence, low power macro verification solutions were not always intuitive, portable or standard. This paper distinctively studies the inherent integration features of soft & hard macros that are inevitable for low power designs today. This has been done by thoroughly identifying the semantic gaps between physical interpretations of macros with their low power orientations. With real design examples, we provided simple and manageable macro verification solutions that are portable, comply with UPF 3.1 standards and reusable in consecutive projects. This will also address verification challenges between flat frontend simulation flows to the hierarchical backend flows. Our motivation is to create a complete low power integration and verification solution for soft & hard macros that will benefit the design, verification, integration, implementation, as well IP vendor industries.**

I. INTRODUCTION

Macros (or IPs) like memories, IOs, PHYs etc. in general, play crucial roles in developing every chip design today. There are soft (synthesizable RTL) and hard (already synthesized, sometimes placed & routed) macros that are generally available for integration in system level designs, like SoCs. In the UPF based low power design, verification and implementation flow (DVIF), macros introduces additional layers of complexities. Specifically, this is because of inadequate knowledge about the identification of macros in UPF boundary concepts and then mapping of additional power artifacts — like driver or receiver supplies, power strategies, terminal boundaries etc. — around them.

On one side, this was due to the lack of UPF standards that will help to identify and map all the power artifacts (ranging from power domain to different power strategies) around the macros. On the other hand, macro integration to the entire design and subsequently conduct power aware (PA) verification lacks in a coordination of knowledge from the physical implementation perspective.

In general, soft macro are part of a larger RTL subtree before implementation. But the fundamental problem arises when they are hardened (synthesized). A standalone self-contained UPF is mandatory for implementation to accurately model the outside environment view of the macro based on the internal power supplies. Because implementation is hierarchical but verification is required in full SoC level as flat view. This may expose conflicting views for implementation and verification.

On the other hand, hard macros during implementation - they usually 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 partial implementation of power architecture and interface characteristics like related supply on logic ports, pg pins, etc. Obviously integration and verification requires adequate information on power architecture interface like driver & receiver supply, missing internal power states etc.

In order to overcome these semantic gaps between physical interpretations as well potential verification and implementation difference of macros with their low power orientations, let us first foster the physical interpretation of the macros. That is we will first investigate - what are the minimum set of boundary parameters that are mandatory to integrate and verify macros with the entire system level design. Eventually we will explore how these verified macros can be reused with new system integration concepts and finally and propose a pragmatic macro integration with PA verification solution.

Gracefully soft macros comes only with power management constraints while hard macros comes with pre-defined UPF and both are pre-verified at IP (block) level. Hence (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.

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 (discussed in detail in Section IV, Table 5).

B. Organization of this Paper:

This paper is organized in the following structure. In section II and III - we have conducted exhaustive studies to identify every possible contributors that forms the foundation of low power macros. These sections also explains the methodologies with specific examples that are defined in the latest UPF 3.1 [5], as well that were missing until the UPF 3.0 language reference manual (LRM) [4]. The next section IV explains the predominant factors that govern macro verification, integration and reuse. The final Sections V draws the conclusion. The references are shown at the end.

II. Low Power Perspective of Soft Macros

Soft macros are generally designed in synthesizable RTL. In UPF perspective, a soft macro model is represented in RTL and associated with the UPF attribute {**UPF_is_soft_macro TRUE**}, which is the soft macro identification directive used in DVIF for verification, integration, soft macro hardening etc. Furthermore, the UPF accompanied with a soft macro must be complete within its entity, i.e. a self-contained UPF that will define its own top level power domain with **create_power_domain -elements {.}** commands. It's obvious for the system integrator to ensure that the macro is connected to proper supplies and that the boundary of IP are properly protected with respect to the parent and between other power domains that have signal crossing. The protection may specify isolation strategies with specific clamping requirements. It may also specify retention hence it is require to ensure backup supplies for the critical registers to retain the data during the power down. In addition, IP integrator must ensure that these constraints (isolation with clamp value, retention registers with etc.) are not violated when the soft macro is integrated into the entire system and configured according to their requirement. For verification, the soft macro power intents or UPF should be complete, independent and must not reference any object to its parent scope so that the verification tools can readily validate and catch any scenarios that fail to honor any of these constraints.

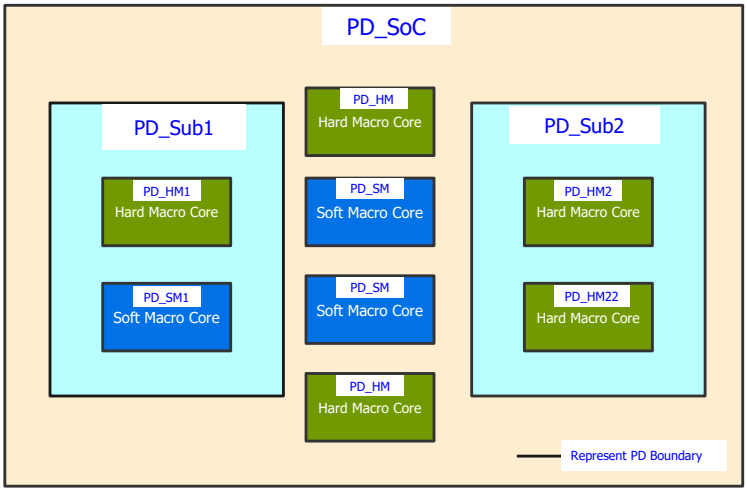

Figure 1 Standard SoC with multiple Hard and Soft Macro Core Integrated

Also the primary IO ports of the macro will not rely on the visibility of real driver and receiver supplies. That is, when driver and receiver supply attributes are present, IO ports will be annotated with these attributed supplies accordingly. However when these supply attributes are absent, the driver and receiver supply will reference to the primary supply set of its own top level power domain. It is also important to know that the driver and receiver supply contexts for soft macro IO ports reveal that they are regarded as the terminal points or terminal boundaries with respect to the ancestor power domain instances for all verification and implementation tools. Such boundary conditions obviously paved the way to reuse same macros over and over again across multiple projects. The following two Tables 1 & 2 shows different ways to construct power management schemes for soft macros.

Table 1 Modeling UPF Constraint for Soft Macro

## Soft Macro UPF: soft_macro.upf ##	## SoC Level UPF SoC.upf ##
create_power_domain PD_Smacro \ -elements {.} \ -supply { cpu_ss } \ -supply { mem_ss } \ **-atomic** **#Isolation Constraints** set_port_attributes \ -ports $PORTS \ -clamp_value 1 **#Retention Constraints** set_retention_elements critical_regs \ -elements { reg_a reg_b } \ -retention_purpose required **#... Other Constraints ...**	**#.... Other UPF Commands ...** # Load UPF of Soft IP load_upf **soft_macro.upf** \ -scope softIPinst **# Connect Supplies** associate_supply_set pd_SoC.primary \ -handle softIPinst/PD_Smacro.cpu_ss associate_supply_set pd_SoC.mem_ss \ -handle softIPinst/PD_Smacro.mem_ss **# Update Supply Constraints** add_power_state pd_SoC.primary .. **# Connect Logic Controls** connect_logic_net ... **#.... Other UPF Commands ...**

It is important to observe here that the power domain of soft macro created with atomicity (with **-atomic** options or **create_power_domain –elements {.}** that contains its own top) which signifies that this domain can be merged but cannot be split during implementation.

On the other hand, a power model defined within **begin ~ end_power_model** or **define_power_model {}** and instantiated in a design with {**UPF_is_soft_macro TRUE**} may also represent a soft macro. The UPF intent within this power model describes the intent of the soft macro hardening process. So, based on the UPF power domain confinement, power model and hardening process, the RTL instance of the macro belongs to the extent of the parent power domain as a hard boundary.

Table 2 Modeling Soft Macro and Its UPF

HDL Design /Instantiation	Macro Model and UPF with Constraint

library ieee;
use ieee.std_logic_1164.all;
entity **MemCtrl** is
port (i1 : in std_logic;
i2 : in std_logic;
i3 : in std_logic;
i4 : in std_logic;
o1 : out std_logic;
o2 : out std_logic);
end MemCtrl;
architecture MemCtrl_arch of MemCtrl is
signal vddA, vddB, vssA, vssB, C : std_logic;
begin
o1 <= i4 and i3;
o2 <= i4 and i3;
end architecture MemCtrl_arch;
	set_port_attributes -ports {W} -clamp_value 0
	define_power_model upf_macroC -for {**MemCtrl**} {
	create_power_domain PD2 -elements {.} \
	-supply { SSAH } -supply { SSBH } -supply { SSBH_SW }
	#Associate the interface supplies to boundary supply ports or
	#Internally generated supplies
	create_supply_set PD2.SSAH -function { power vddA } \
	-function { ground vssA } -update
	create_supply_set PD2.SSBH -function { power vddB } \
	-function { ground vssB } -update
	#Special handle for internally generated supply
	create_supply_net vddB_int
	create_supply_set PD2.SSBH_SW -function {power vddB_int} \
	-function {ground vssB} -update
	set_design_attributes -models MemCtrl -is_soft_macro TRUE;
	}

# Instantiation **MemCtrl** Y5(W, X, Y, Z, Z1, Z2)	**# Binds Soft Macro in the design and connects the interface supply** **# set handles of a previously loaded power model.** load_upf ./MemCtrl.upf apply_power_model upf_macroC -supply_map { { PD2.SSAH PD1.SSAH } {PD2.SSBH PD1.SSBH} }

It is important to note that UPF 3.1 indicates that UPF must be self-contained for macros. Hence it is also essential to know that the soft macro instance is considered to have a terminal boundary that restricts the scope of the object. That is why power intent objects expressed in an ancestor like global supply sets are not available to the block and therefore the power intent must be supplied explicitly even for soft macros.

Figure 2 shows the soft macro instantiated in '**tb.top_inst.I2.X5.Y5**' in hierarchal windows on upper left, the HDL code in RTL shown in upper middle, corresponding atomic power domain PD2 is shown in upper right, the reports from simulation results are shown in lower left (where simulation consider the power model as soft macro cell) and power domain status (where corruption status is shown unknown since macro didn't specify any power states) is shown in lower right windows.

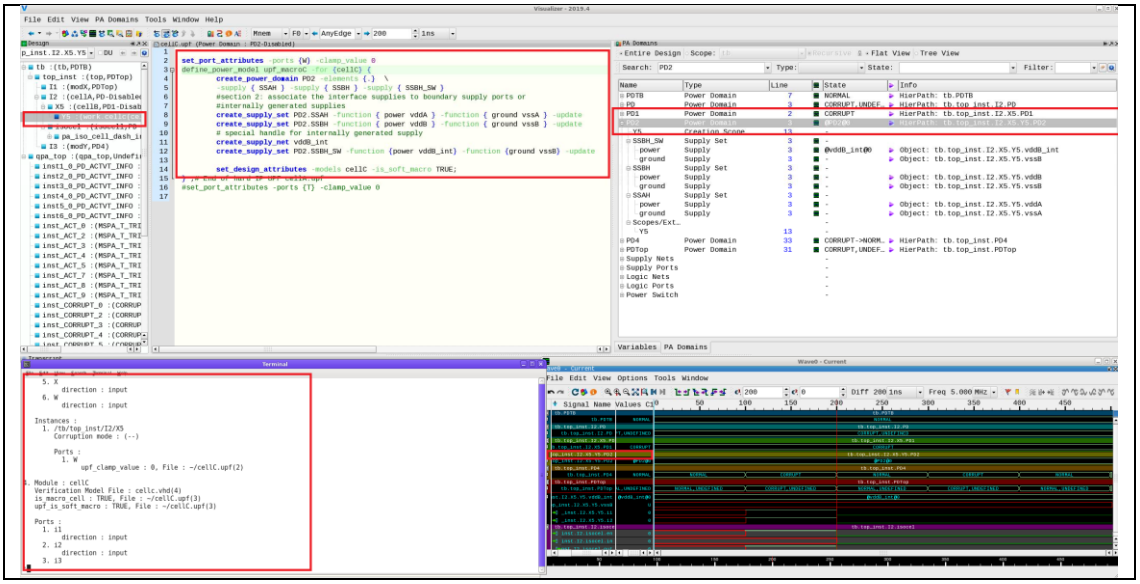

Figure 2 Soft Macro Power Model Verification

III. Low Power Perspective of Hard Macros

On the contrary, the hard macros are generally designed in a full custom approach in the form of hardware IP. These are the block level designs which are often silicon proven and optimized for power or area or timing. From the physical implementation standpoint, only the supplies, IO pins and ports of hard macros are visible or accessible for integration and verification. In the UPF perspective, a hard macro is an IP block instantiated in the design with the UPF HDL attribute {**UPF_is_hard_macro TRUE**} or Liberty cell library attribute `<is_macro_cell:true;>`, which are hard macro identification directives for verification and implementation tools. A hard macro defined using the Liberty `<is_macro_cell:true;>` attribute will implicitly set the {**UPF_is_hard_macro TRUE**} on the HDL model.

The power management intents or UPF for any hard macro contains the information of internal-external or driver-receiver supplies, related supplies for its boundary IO ports, isolation, level-shifter, repeater cells that are already implemented within the macro or required on its boundary and modes of power states and/or voltage values. It is also expected that the UPF is already verified in macro or block level. However, just like soft macros, it is equally important to verify the accompanying UPF with the entire system once the hard macro is integrated. The SoC integrator ensures that these macros are properly connected to appropriate system level supplies and the boundary are properly protected with respect to the entire system where the macro is placed. This will ensure that verification tools can validate the power management in the SoC environment and can catch issues related to macro at early in the design flow. The following two Tables 3 &4 shows different ways to construct power management schemes for hard macros.

Table 3 Modeling UPF for Hard Macro

```
## Hard Macro UPF: hard_macro.upf ##
#Power Domains for Hard Macro
create_power_domain PD_ Hmacro \
  -include_scope \
  -supply { backup_ssh } \
  -supply { primary }
#Related Supply Constraints
set_port_attributes -domain PD_Hmacro \
  -applies_to outputs \
  -driver_supply PD_Hmacro.primary
set_port_attributes -ports portA \
  -driver_supply PD_Hmacro.backup_ssh
set_port_attributes -domain PD_Hmacro \
  -applies_to inputs \
  -receiver_supply PD_Hmacro.primary
#Internal switchable supply
create_power_switch ...
# Isolation/level shifter and retention cells
  set_isolation ...
  set_level_shifter ...
  set_retention ...
# System states for hard macro
```

```
## SoC Level UPF SoC.upf ##
#.... Other UPF Commands ...
# Load Hard Macro UPF
load_upf hard_macro.upf \
 -scope
# Connect Supplies
associate_supply_set PD_SoC.primary \
-handle Hmacroinst/PD_Hmacro.primary
associate_supply_set PD_SoC.backup \
-handle Hmacroinst/PD_Hmacro.backup
# Update Supply Constraints
add_power_state PD_SoC.primary ..
# Connect Logic Controls
connect_logic_net ...
#.... Other UPF Commands ...
```

add_power_state PD_Hmacro -state {ON -logic_expr {primary == ON}}…	

Table 4 Modeling Hard Macro and Its UPF

HDL Design /Instantiation	Macro Model and UPF with Constraint

# Hard Macro 	
module **Hmacro**(W, X, Y, Z, clk, Z_OUT);
parameter WIDTH = 2;
input W, X, Y, Z, clk;
wire vddA, vddB, vssA, vssB;
wire Z11, Z22;
output reg [WIDTH-1:0] Z_OUT;
# Internal isolation 	
isocell isocel(iso_x, W, X);
# Other macro instantiation 	
cellB X5(W, iso_x, Y, Z, Z11, Z22);
# Logic and functionality 	
and a1(Z11,Z, iso_x);
and a2(Z22,Z, iso_x);
 	
always @(posedge clk) begin
Z_OUT[0] <= Z11;
Z_OUT[1] <= Z22;
# Other Conditions… 	
end
endmodule
	# Top level PD
	create_power_domain PDTop -elements {.} -supply { SSH1 }
	# Loading Hardmacro
	load_upf ./Hmacro.upf
	# defining the interfaces of hard macro power model
	define_power_model upf_macroA -for {**Hmacro**} {
	create_power_domain PD -elements {.} \
	-supply { primary } -supply { SSBH } -supply { SSBH_SW }
	# Liberty attribute
	set_design_attributes -models **Hmacro** -is_hard_macro TRUE
	# Associate interface supplies to boundary supply ports
	create_supply_set PD.primary -function { power vddA }
	-function { ground vssA } -update
	create_supply_set PD.SSBH -function { power vddB }
	-function { ground vssB } -update
	# Handle for internally generated supply
	create_supply_net vddB_int
	create_supply_set PD.SSBH_SW -function {power vddB_int}
	-function {ground vssB} -update
	create_power_switch internal_sw -output_supply_port {out PD.SSBH_SW.power} \
	-input_supply_port {in vddB} -control_port {ctrl Y} -on_state { ON in !ctrl }
	# Define IO port interface supply set handle associations
	set_port_attributes -ports { W X } -receiver_supply PD.primary
	set_port_attributes -ports { Z_OUT[1] } -driver_supply PD.SSBH
	# **Define power states for interface supply set handles**
	add_power_state PD.primary -supply\
	-state {ON -simstate NORMAL \
	-supply_expr {power == { FULL_ON, 0.7, 0.9} && ground == { FULL_ON, 0}}}\
	} ……
	#Define system SoC level power states of the hard macro
	add_power_state PD -domain\
	-state {S1 -logic_expr { primary == ON && SSBH == ON && SSBH_SW == ON }} \ …
	# Define hard macro level isolation constraints
	set_port_attributes -ports {W Z} -clamp_value 0
	set_port_attributes -ports {Y} -clamp_value 0
	} # **end_power_model # End of hard IP UPF Hmacro.upf**
	# Binds power models to instances in the design
	apply_power_model upf_macroA -elements {**Hmacroinst**} -supply_map { { PD.primary PDTop.SSH1 } {PD.SSBH PDTop.SSH2} }

# Hard macro instantiation **Hmacro** **Hmacroinst**(in3, o1_wire, in2, o2_wire, clk, {out1_wire, out2_wire});	

# Liberty attributes in UPF, HDL, .lib 	
 	
# Liberty attribute in UPF 	
set_design_attributes -models **Hmacro** -is_hard_macro TRUE
 	
# Liberty attribute in HDL 	
(*UPF_is_hard_macro="TRUE"*) module **Hmacro**
 	
# Liberty attribute in .lib 	
library(pll_lib) {
cell (**Hmacro**) {
is_macro_cell : true;
…
}	

Noticeably, hard macros can be available in an HDL behavioral model for verification or as a Liberty cell for implementation. Here we have shown the use case of hard macro that is designed in behavioral HDL but hard macro could also be instantiated as synthesized gate-level blocks. During verification, it is important to remember that the hard macro is just a black box to the rest of the design. Only the interface (i.e. logic pins or IO ports and related supplies) of the models are visible to the parent context. As a result, PA verification of hard macros have several *verification use case*s or *use models* based on how a set of macro HDL models, Liberty cells and UPF objects are available in any particular verification environment. The possible and acceptable combination of these components for any *verification use model* are listed below.

List 1: Different PA *Verification Use Models* of Hard Macros

(a) HDL macro behavioral model or gate-level netlist, Liberty cell library and self-contained UPF with macro's own top level power domain defined with **create_power_domain -elements {.}**.

(b) HDL macro behavioral model or gate-level netlist, Liberty cell library and UPF without macro's own top level power domain defined.

(c) HDL macro behavioral model or gate-level netlist and Liberty cell library.

To note, all the Liberty attributes, like <is_macro_cell:true;>, and PG-pins, i.e. the related or driver-receiver supplies etc., can readily be derived from the combination of HDL and UPF attributes and commands. So, a Liberty cell library is not mandatory for (a) and (b). Although each set of the above *use models* seems different, all the

sets tend to extract the exact same or similar information for PA verification, like power domain boundary, terminal conditions, related or driver-receiver supplies etc.

Let us consider the case (a), which actually started out being a soft macro and later got hardened through the implementation process. Here the same self-contained UPF with its own top level power domain is used for both verification and implementation. It is important to notice here that the UPF in both cases of soft and hard macros contains **create_power_domain -elements {.}**. So, similar to the soft macros, the behavioral HDL or gate-level netlist of hard macro instance belongs to the scope of its own parent power domain. And since the macro is a black box to the rest of the design, every IO port in the hard macro is considered as a terminal boundary.

For (b), the accompanying UPF is defined without its own top level power domain, so it's expected that the UPF at least specify the driver and receiver supply attributes for the macro ports. Here the hard macro instance or any of its IO ports do not belong to the extents of any parent power domain. This analogy equally applies for (c) as well. Physical interpretation and black box concepts of hard macros made it evident for (b) and (c) that regardless of whether the related supplies of these IO ports are different or not, all the IO ports of a hard macro will be regarded as terminal boundaries. Also just like soft macros, a power model defined within **begin/end_power_model** but instantiated in a design with {**UPF_is_hard_macro TRUE**} may also represent a hard macro. The UPF intent within this power model describes the intent in such a way that the macro has already been hardened through the implementation process.

One big concern still remains regarding the extents of a power domain for the instances of hard macros in the case of (b), where the parent or ancestor scope is undefined, and for (c), where the UPF itself is absent. For such cases, if we query the extent of the power domain of the macro instance, it will be included in its ancestor power domain scope. Consequently, any unconnected PG-pins of the macro will be then associated to the ancestor instance power domain primary supply. However, from a hard macro physical interpretation perspective, it's appropriate to consider such cases of hard macros to belong to an independent power domain that will be powered by the primary supply of the ancestor domain. This separate power domain ensures that all the ports of the macro will be considered as power domain boundary ports with respect to the ancestor instance power domain.

Just like soft macro, obviously such boundary conditions and constraints allows to reuse the verified hard macros across different projects. The following Figure 3 shows the hard macro verification environment. In this diagram, the hard macro named '**Hmacro**' is instantiated in '**tb.top_inst.Hmacroinst**' in hierarchal windows on upper left, the UPF 3.1 power model code is shown in upper middle, corresponding power domain PD is shown in upper right, the liberty file present in the verification is shown on extreme upper right. The reports from simulation results are shown in lower left (where simulation consider the power model as hard macro cell) and power domain status, variables (IO, reg, wires etc.) are shown in lower right windows.

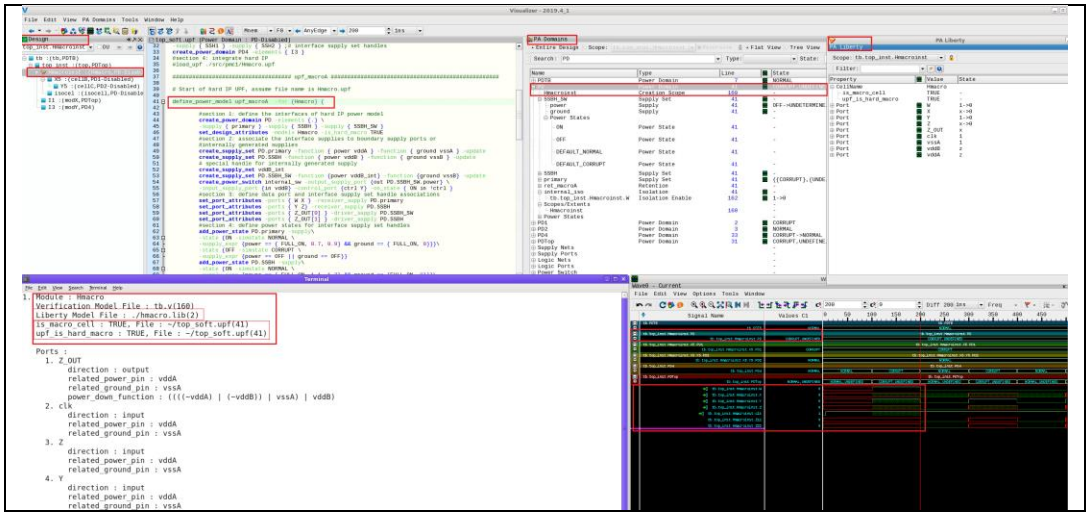

Figure 3 Hard Macro Power Model Verification

IV. Predominant Factors that Govern Macro Verification and Integration

Even though we have explained the PA 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 DVIF. Table 5 summarize the complete lists of all the predominant factors that affects and governs macro integration and verification.

Table 5 lists of all the predominant factors that affects and governs macro integration and verification

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 expectations,	Integration, Verification
Simulation state behavior,	Verification
Corruption semantics etc.	Verification
Flat Vs Hierarchical design (reason of having terminal point)	Integration, Verification, Implementation

The fundamental constructions of standard SoC design usually proceeds in hierarchal manner where the 'design cores' comes first and 'sub-system' integration follows these cores (refer to Section II, Figure 1). Depending on the specification and requirements, the 'cores' could be designed as a hard or soft macros. Obviously the power intent or UPF development for SoCs also occurs in hierarchical manner. The hard or soft macro boundaries are typically power domain boundaries as explained earlier in section II & III, allows synthesis to generate 'core' level designs (also known as soft macro hardening) for soft macros and black box synthesis for hard macros.

However, it is important to note that there are distinct requirement differences for synthesis and simulation (or verification) for these macros. Specifically for hard macros, the synthesis requires a power model or Liberty library to allow hierarchical synthesis at sub systems or SoC level. But for simulations, instead of just power model or Liberty libraries, complete UPF including design scope, power states for interface supply set handles, boundary strategies, maps for the interface supply sets of the hard macro is required to connect with the supply sets in the loaded scope. Hence it clarifies another aspects of design observability expressed in terms of fully flattened and hierarchical views that we identified as an issue and stated in problem formulation in Section I of this paper. The synthesis-simulations perspective of UPF for macros clarifies that simulation flows always consider flattened designs, whereas back end implementation flows typically requires hierarchical views.

This is very crucial aspect of UPF due to the assumptions involved in defining driver supply and receiver supply for macros, specifically:

- When synthesized at core level – it require to model the internal conditions of the macro - receivers for inputs, drivers for outputs.
- When integrating the macro in higher implementation level – it require to model the internal conditions of the macro - receivers for inputs, drivers for outputs
- When verifying the macro – no requirements for interface conditions as all driver and receiver information is available in the design and UPF.

So these clarifies the reason of having different views for synthesis and simulations for macros.

In addition, the terminal boundary – which is typically the power domain boundary for both hard and soft macros, plays significant roles in PA verification. Specifically for developing dynamic custom verification or populating additional UPF based on **find_object** tracing and traversing through HDL for objects. For example, **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. It is also important to remember that global supply net declaration and location fanout for UPF strategies are not applicable for terminal boundaries.

V. Concluding Remarks

In this paper we have taken an exhaustive initiatives to identify and resolve the fundamental problems of integrating, hardening and verifying soft & hard macros in larger systems with power aware DVIF perspective. In order to sustain a pragmatic approach to reduces the complexities of reusing power aware macros – we identified that for both the macros –a self-contained UPF is essential. For soft macro, atomicity of the power domain, internal retention & external boundary strategies, legal power states and sequencing between them are crucial. For hard macros, specification about related supplies of boundary ports, external and internal supplies and their characteristics, internal power states of power nets and switches, internal & external boundary strategies with supply information and system level power states are mandatory.

ACKNOWLEDGEMENT

The authors would like to express sincere gratitude to the reviewers from Mentor Graphics Corp. 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] Progyna Khondkar, et al., "Low Power Coverage: The Missing Piece in Dynamic Simulation", February March, DVCon 2018.

[3] Progyna Khondkar, et al., "Free Yourself from the Tyranny of Power State Tables with Incrementally Refinable UPF", February March, DVCon 2017.

[4] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801-2015, 5 December 2015.

[5] 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 (Revision of IEEE Std. 1801-2013).