2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

# *Power models* & Terminal Boundary: Get your IP Ready for Low Power

Progyna K., William W., Phil G., Brandon S.

infineon

cadence®

accellera SYSTEMS INITIATIVE

# Outline & Contribution of this Research

- This paper establish the transparent relations among
  - UPF power model and terminal boundary
- Identified and reinforced a complete perception of
  - Low power Soft and hard macro design, verification, implementation, and integration, as well reuse environment.
- In order to do so,
  - First explain what soft and hard macros are and how they appear
    - In real low power design verification, integration and implementation environment.
  - Then explain how self-containment of 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.

# Agenda

- What is UPF & UPF Power Model
- What is Soft & Hard Macro
- What is Terminal Boundary
- What are the Problems of Low Power IP Designs
- How to resolve low power IP design verification and Integration
- Methodologies for Low Power IP Design, Verification & Integration
- Case Studies
- Concluding remarks

# UPF

- The Unified Power Format (UPF) also known as IEEE-1801,
  - 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 also provide 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.

# 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*'
  - Combined 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 model*s
  - 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.

- The *UPF power model* is the concept of self-containment of power intent for soft and hard macros.

# 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, following are mandatory for soft-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.

- 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).

# 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 commands are not allowed
  - 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 prohibited too.

- 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.

# Problems with Low Power IPs

- Inconsistent definition or name of low power Soft and Hard macros
  - Incomplete or not distinctly understandable requirements for UPF
  - Verification, Integration requirements are unknown.
- 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.
- Identify and establish a complete perception of soft and hard macro
  - verification,
  - integration and
  - reuse environment.
- We exercised all the predominant factors that govern
  - The simple and manageable macro verification and integration (possibly reuse) solutions.

# Self-Containment UPF for Macros

- 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.
- Note:
  - A power model that is not referenced by an apply_power_model command
  - Will not have any impact on the power intent of the design.

- 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 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’

# UPF Power Model?

**“UPF power model” can be used to represent one of the following:**

- For a hard macro,
  - The power model defines the attribute ‘UPF_is_hard_macro TRUE’
  - The UPF commands within a power model describe power intent that **has already been implemented**
    - That’s the reason for hard macro “a self-contained UPF **may not be mandatory** but usually ships with ‘UPF_is_hard_macro’ attribute set to true.”
  - So, verification tool can get power intent info or validate it from liberty or combinedly both liberty and power model.
- 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 a soft macro,
  - The power model define the attribute UPF_is_soft_macro TRUE.
  - The UPF commands within the power model describe power intent that **remains to be implemented**.
- A Soft macro,
  - UPF power model **is mandatory**.
  - The requirement can be readily verified according to the UPF 3.1 as follows.
    - It shall be an error if the UPF specified for a soft macro is not self-contained.

# Self-Containment UPF for Macros

**Top Level design/dut, macro and instantiation of macro**

```
module wrap_tb();
top dut (clk,reset,inp,outp,buff);
endmodule

module top (clk,reset,inp,outp,buff);
..
supply_net_type vdd,vss;
cellA   I1 (vdd,vss,clk,reset,inp,temp);
endmodule

module cellA (vdd,vss,clk,reset,inp,outp);
<other logics>
endmodule
```

**Top Level UPF top.upf**

```
# 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 } }
```

***UPF power model* macro.upf**

```
# 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}) && (ground == {FULL_ON,0.0})} -simstate NORMAL}\
-state {NON -supply_expr {(power == {OFF}) && (ground == {OFF,0.0})} -simstate CORRUPT}

# Define Internal ISO, RET, Power Switch strategies
…

}
#end of encapsulation
```

# Why Self-containment?

- Todays SoC, consists of
  - several macros and other synthesizable (or implementable) logic blocks.
    - While implementing such larger system
      - Requirements may come to implement an instance separately from the top-level scope
        - The intention is to integrate this block back into the system later in the flow.
  - This flow style is often referred to as a bottom-up flow.
    - Bottom-up flow requires certain considerations regarding UPF partitioning.

- In bottom-up integration
  - The implementation of a lower-level instance done without parent scope
    - 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.
- Therefore, the block UPF power intent must be self-contained;

# Why Self-containment?

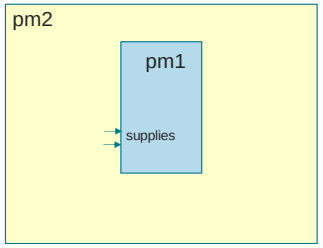
- A SoC may contain instances
    - That already been implemented or
    - instances that will be implemented separately.
- These instances in a bottom-up flow need to be defined as macros.
    - By defining an instance to be macro,
        - The evaluation of certain UPF power intent commands are affected
            - Because a macro forms a terminal boundary.
- When 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.

# Nested Power Model

- Nested power model are also allowed
  - one power model applied to a given instance
    - may apply another power model to a descendant instance.
      - e.g. applying ‘apply_power_model’ inside another power model
  - This eventually will also require
    - Multiple load_upf -scope inside the power model for the nested apply_power_models.

**Example of Nested Power Model:**

```
define_power_model pm1 ... {
....
}
        define_power_model pm2 ...  { ....
         apply_power_model pm1 ...  ...
    }
```



# Macro Integration for Verification

- How to hookup macro for verification or integration on to larger system?
- Supply and logic connections are utilized to
  - Hookup the macro through UPF power model
    - For verification, and eventually integration on to larger SoC contexts.
- For supply
  - ‘apply_power_model’ -supply_map
- For Logic
  - connect_logic_net current_scope_logic_supply_net –ports {instance_scope_logic_supply_port}
  - connect_supply_net current_scope_logic_supply_netb–ports {instance_scope_logic_supply_port}

# Macro Integration for Verification

- For supply connection - UPF allows to map
  - A supply set in parent scope to a supply set in the power model.
  - Also allows mapping a parent supply set to 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.

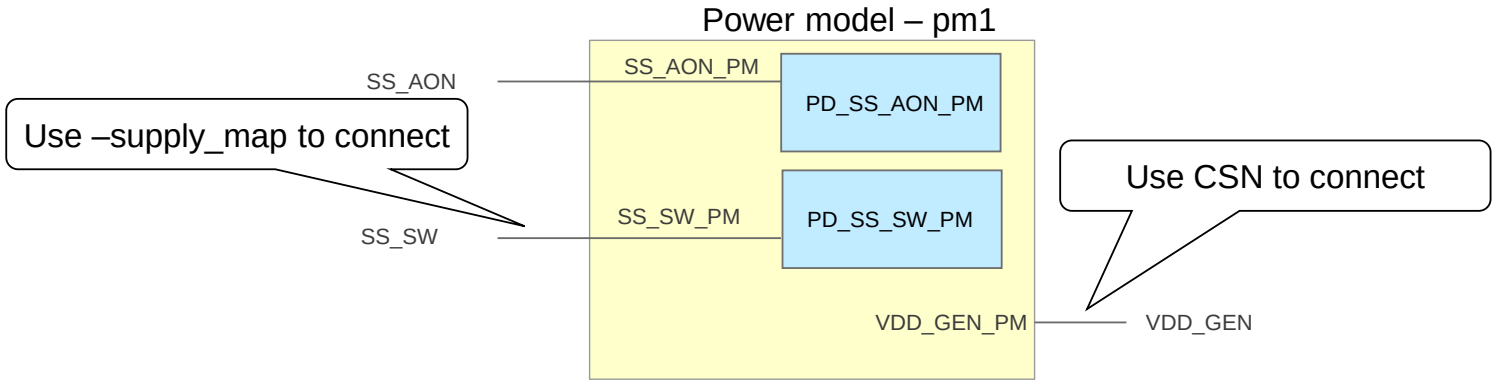
- For logic connection – UPF allows to connect
  - From logic/supply ports of the power model to logic/supply nets in parent contexts
  - Each pair in the -port_map option implies either
    - A connect_logic_net command  or
    - A connect_supply_net command
  - Depending on weather it's a logic connection or a supply connection.

# Macro Integration for Verification

**UPF power model and Parent supply association**

```
# Power model and relevant UPF commands for inside the macro
begin_power_model pm1 {
  <all upf commands that apply inside of the power model>
}
# 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} …}
```




- apply_power_model 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.

# Macro Integration for Verification

***UPF power model* and Logic/Supply nets**

```
# 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_set SS_VDD -function { power VDD }        -function { ground VSS }
create_supply_set SS_DVDD -function { power DVDD } -function { ground DVSS }
# Define Power Domains
create_power_domain pd_test_macro -elements {.}\
      -supply {first SS_VDD} \
      -supply {second SS_DVDD} \
      -supply {third SS_DVDDLO}
associate_supply_set SS_VDD -handle pd_test_macro.primary
```

***UPF power model* and Logic/Supply nets**

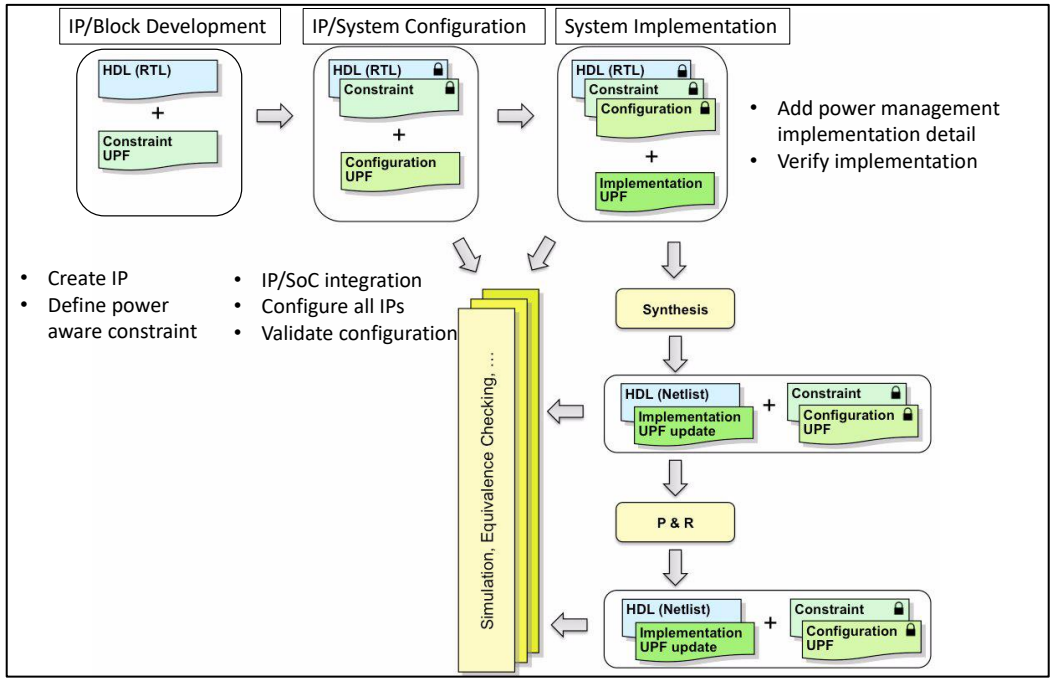
```
# Parent Contexts
create_supply_set SWCoreSupply \
      -function [list power VDD_SW_top] -function [list ground VSS_SW_top]
# Parents Domain
create_power_domain PD_Parents -elements {.} \
      -supply {first SWCoreSupply } \
      -supply {second SWIOSupply } \
associate_supply_set SWCoreSupply -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} \
```

- -port_map shows the interface logic or supply ports of the instance scope
  - That is macro in define_power_model connects with logic or supply nets
  - In the current scope where apply_power_model is applied

# Successive Refinement & Macros

- As the design progress through phases
  - Obviously UPF information grows for each of these phases.
- Hard macros are typically already implemented and
  - Hence not actually relevant for this methodology,
  - Although its relevant for verification.

- For Soft macro verification and implementation
  - In bottom-up integration, successive refinement can play significant role
  - In such flow, soft macro will require
    - Constraint UPF
    - Configuration UPF
    - Implementation UPF all together for verification
  - In addition
    - liberty will be required for implementation
    - (i.e. synthesis/hardening after verification at RTL).




Courtesy: IEEE Std 1801™-2018

# Successive Refinement & Macros

- The constraint, configuration and implementation UPF for IPs
  - Remains within “UPF power model” through define_power_model commands.
- Soft macros follows
  - self-contained UPF semantics and
  - Act as terminal boundary.
- Hence successive refinement flow is ideal for bottom-up integration
  - However, successive refinement flows have known limitations
    - Like soft macros are not refinable.
    - e.g., Optimize or remove redundant strategies or
    - Change / modify locations of strategies inside of macros
  - Even the implementation UPF may not be altered
    - For a different technology library mapping.

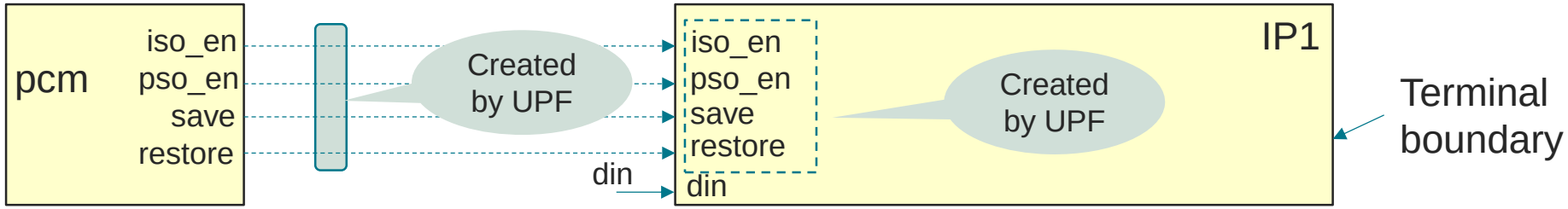
# Limitations of UPF Power Model

- Certain limitations of verifying soft macros at block or system level
  - e.g., due to terminal boundary restrictions,
  - User have no access and control of the power enable signals
  - Even UPF have mechanism to create/connect logic nets & ports in a scope
    - Through create_logic_net/port and connect_logic_net_port
  - But such provisions are restricted by terminal boundary.
- How to access?
  - Different power control signal within a terminal boundary as input ports only
- How to associate?
  - A driver supply with a logic port to confirm that the correct supply is ON for the control enable signals.

# Limitations of UPF Power Model

**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
```




- Ideally set_port_attribute command for logic port and HDL port have no functional difference.
  - But LRM restricts the extension of such input power enable ports with soft-macro
    - And associate them with relevant driver supplies.
- Note: find_object will still not work for accessing such logic ports unless -traverse_macro is specified

# Boundary Condition

- Bottom-up integration & implementation
  - Most productive choice for SoC design-verification

- UPF portioning is obvious- specifically for implementation
  - A lower-level instance needs to be done without the parent scope.
    - So, block-level UPF (power intent) must be self-contained.
    - 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 every aspect of larger SoC contexts are not available,
  - It is obvious to make some consideration regarding the boundary conditions for the model (IP or macros)

# Boundary Condition

- Whether dealing with hard or soft macros, the parent context of the UPF
  - Specifies only those conditions as seen from outside of these macros
  - (where UPF power model of macros are loaded and apply_power_model is used).

- This is why driver/receiver_supply are mandatory for soft & hard macros
  - These specify appropriate from the outside of these macros (parent contexts).
    - driver_supply for output and
    - Receiver_supply for input ports
  - And exact opposite from inside of these macros (self-contexts).
    - driver_supply for input and
    - receiver_supply for output

  - NOTE: Inside declaration are optional for hard macros.
    - But its obligatory to verify when or if they are present in accompanied UPF power model.

# Boundary Condition

| 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 |
| *driver_supply* for output and *receiver_supply* for input from parent-contexts |
| *driver_supply* for input and *receiver_supply* 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-shifer 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 |

# Macro Verification, Integration & Implementation

- Methodological aspects that governs
  - 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 state expectations | Integration, Verification |
| Simulation state behavior | Verification |
| Corruption semantics etc. | Verification |
| Flat vs Hierarchical design (reason of having terminal points) | Integration, Verification, Implementation |

# Concluding Remarks

- 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 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 (simulation) 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.

- Hope this paper, will serve as reference point to prepare IPs
  - For low power verification, integration as well implementation.