Verification Macros: Maintain the integrity of verifiable IP UPF through integration

Amit Srivastava
Amit.Srivastava@synopsys.com
Synopsys Inc

Shreedhar Ramachandra
Shreedhar.Ramachandra@synopsys.com
Synopsys Inc

***Abstract*: System on Chips (SoCs) are made up of many soft IPs, which are verified as standalone blocks, including the power management architecture. These IPs are shipped with their UPF containing the power intent in a technological independent form. These soft IPs are integrated into a larger block, which goes through the implementation (synthesis and place and route (PnR)) stage. The UPF for these IPs is supplemented with implementation details to synthesize power management architecture efficiently. To achieve a faster turnaround time, the users reuse the validation done for the IPs. However, the current successive refinement methodology defined in the UPF has various limitations concerning pre-verified IPs (PVIPs). The UPF defines soft macros, which are used for bottom-up implementation flows. The soft macros have several restrictions related to implementation control from SoC which creates challenges in using them for PVIPs. As a result, users are forced to adopt intrusive methods of changing the original power intent to make it go through implementation. This seriously affects the verification done at the IP level and forces users to either revalidate it or risk leaking silicon bugs. This paper details a new methodology to enable the successive refinement of PVIPs (Verification Macros). The proposal is under consideration to be included in the next UPF revision.**

I. INTRODUCTION

Today's SoCs are incredibly complex and contain various subsystems, soft IPs, hard IPs, and other logic. They are typically built in a bottom-up approach where the verification and implementation are done at different levels (Figure 1). Often there are scenarios where the verification and implementation boundaries do not align. This poses challenges to applying UPF power intent with consistent semantics.

Soft IPs require standalone exhaustive validation, including the power management architecture. These IPs are called **pre-verified IPs (PVIPs)**. They have their own UPF, which describes the power management architecture used within the IP. This UPF contains a high-level power architecture independent of the target technology and gets reused for different implementations. These IPs are integrated into a larger block along with other logic. The implementation (synthesis and PnR) is performed at the larger block (APR) to achieve the best area and performance. During the implementation at the APR block, the UPF of the PVIP needs to be updated/refined to include additional details required to implement the power management architecture. This contains target technology details, power switches, and additional logic to help implement the power management. The changes done to the UPF during implementation should not affect the functionality and original power intent. The UPF is added with additional details to make the power architecture more efficient and conducive for implementation. The goal is to ensure that the power intent of the PVIP is not fundamentally altered, which would need to be revalidated using simulation.

In this paper, we define a methodology based on the UPF's Successive Refinement methodology that empowers tools to perform automatic checks and provide confidence to the users that the UPF changes to the PVIP are safe and will not require it to be re-verified.

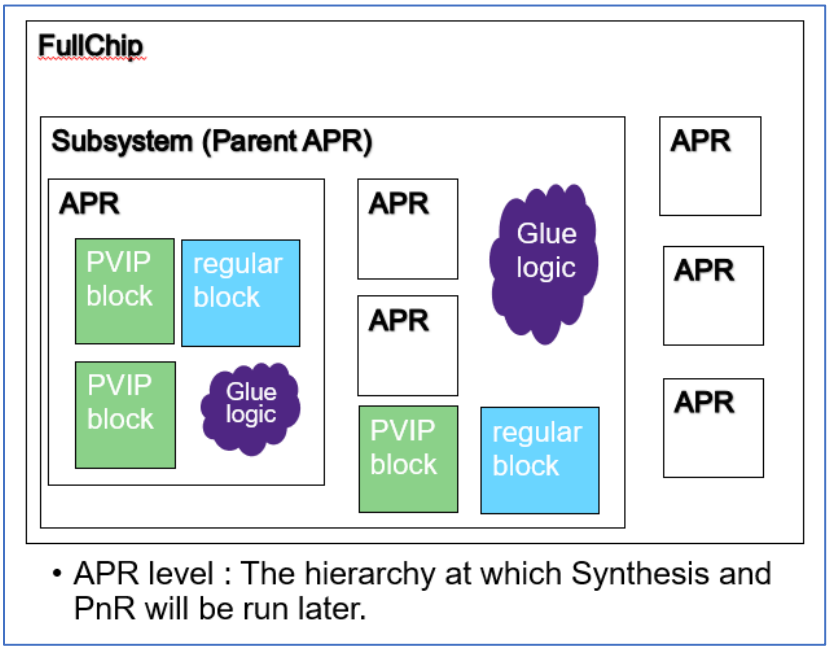

Figure 1 Bottom-up Verification and Implementation flow structure

II. PREVERIFIED SOFT IPS

A common IP integration method used in industry is for the IP to be pre-verified and then implemented in a larger, synthesized context to build an SoC. The soft IP will have a self-contained UPF for power-aware simulation and static checks. In this case, the IP is considered "soft" because it is not pre-hardened by the provider. The SoC will validate the full integration but does not revalidate the IP to what the provider did before delivery to the SoC. In other words, the IP is only revalidated in the context of the larger system and only in the context of the larger system's power flows. For the SoC to optimize timing, power, and area utilization, it may be necessary to modify the UPF of the soft IP. Some aspects of the UPF are not modifiable today without using intrusive methods, such as manually editing the source UPF from the IP provider. When an SoC uses these intrusive methods to modify the IP, the power intent may be fundamentally changed. If the power intent is changed radically, the verification boundary of the IP will be compromised. It is, therefore, essential to allow for modifications to the soft IP UPF to support implementation needs and enable tools to verify that the power intent was not changed in such a way to have compromised the validation done by the IP provider. The next section introduces Successive Refinement, the UPF standard's way of modifying soft IP UPF for implementation needs.

III. SUCCESSIVE REFINEMENT METHODOLOGY

The UPF standard defines an IP reuse methodology called Successive Refinement Methodology. This enables the reuse and progressive refinement of IP power intent when it goes through the design and verification flow. Figure 2 summarizes the UPF successive refinement methodology.

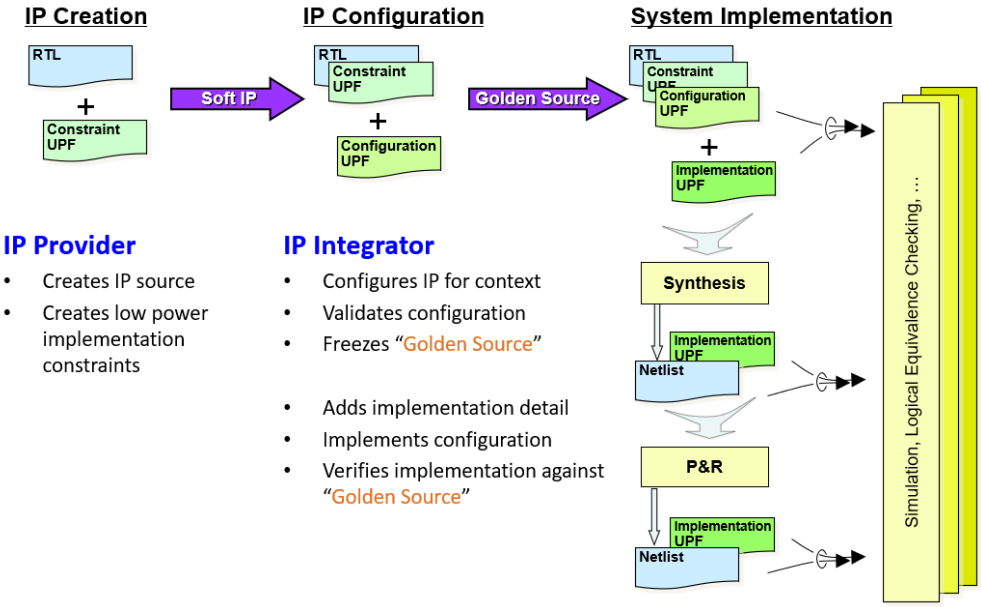

Figure 2 UPF Successive Refinement Methodology

IV. BOTTOM-UP IMPLEMENTATION

In a bottom-up implementation flow, the IP is implemented separately in a bottom-up manner and then assembled in a larger block to create a Subsystem or SoC. In this flow, the verification is typically done at a higher level when there is full system visibility. To achieve faster turnaround times, the verification is performed using the original RTL along with UPF. Since the availability of a larger context can affect the interpretation of UPF, the UPF language allows marking these separately implemented IPs as "Soft Macros". Soft macros enforce specific requirements for the UPF to allow consistent semantics across different tools and verification. The semantics are "Self-contained UPF" and "Terminal Boundary" (Section 4.9.2.2 and 4.9.2.3 in IEEE 1801-2018 [1]). By marking these blocks as soft macros, verification tools can interpret the UPF in the same way as the separate implemented block because the external environment will not affect the power intent within the IP. In the term of UPF LRM, it becomes a terminal boundary.

Figure 3 demonstrates the semantics of soft macros in a bottom-up implementation flow. In the figure, a soft macro IP has two supply pins on the interface. The UPF is written in a way that results in the insertion of isolation cells within the IP from a source powered by one supply to a sink powered by another supply. Since the IP is implemented separately, the isolation cells are present in the netlist. When the IP gets integrated in the larger APR block, there are two separate instances of the IP. For one instance, the two supplies at the interface are shorted together, making the isolation cells redundant. Since the IP is already implemented, verification tools need to ensure that the presence of redundant isolation cells will not affect the IP's functionality. By marking the IPs as soft macro, verification tools can identify the IP boundaries and treat them as terminal boundaries to preserve the redundant isolation cells, even if the supplies are shorted. Note the example of redundant isolation cells is just one scenario. Many such UPF semantics are affected by the environment, so it is important to mark these blocks as soft macros.

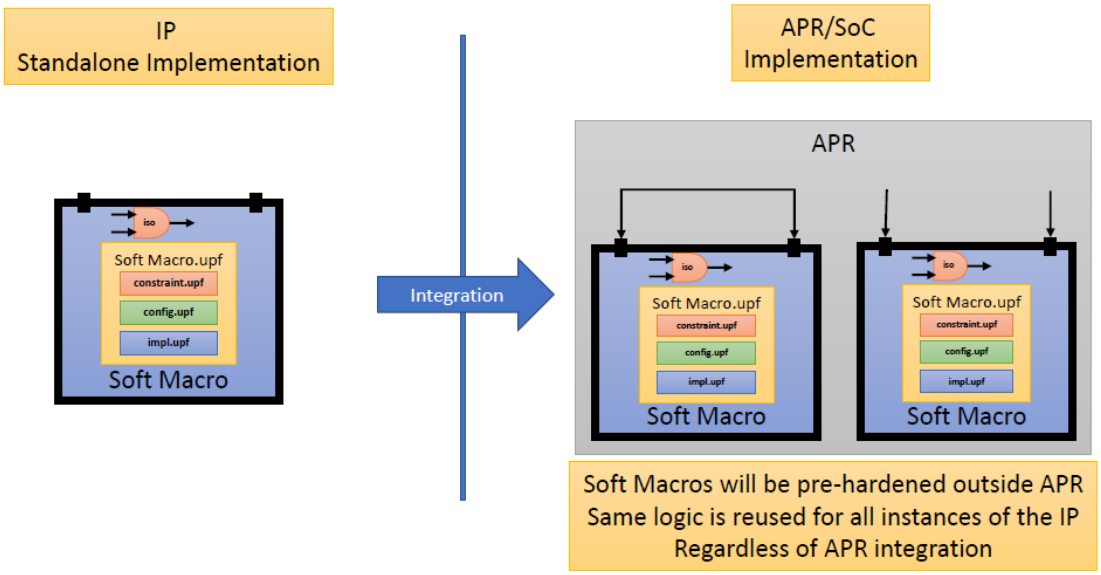

Figure 3 Bottom-up implementation using Soft Macros

V. BOTTOM-UP VERIFICATION

A bottom-up verification flow involves PVIPs. In this flow, the IPs are separately verified but implemented in a larger context. In this flow, implementation tools want a larger context to perform more optimized system implementation. To achieve best timing and area utilization, implementation tools need to add some refinement to the abstract power architecture used during IP validation. However, the refinement should not violate the power intent used during the IP standalone verification. It can optimize away some redundant logic if it is electrically safe to do so.

Figure 4 demonstrates the semantics of bottom-up verification for PVIPs. In this case, the IP has similar power intent as Figure 3. The difference is that IP is validated as a standalone block without any implementation details for power intent. The IP verifies the isolation requirement and ensures that it functions correctly when different supplies power the two supplies of the IP. When the IP gets integrated into the larger system, i.e., APR block, the implementation tool will see that the two IP supplies are shorted. Hence, removing the isolation cells between the ports powered by the respective supplies is safe for better area utilization. It will optimize the redundant isolation cells for one instance but preserve the isolation cell for the instance powered by different supplies. The IP integrator will need to provide an additional UPF containing the details required for implementing the power intent. To preserve the validation already done for the IPs, the additions to UPF should be made in a way that doesn't violate the original power intent of the PVIPs.

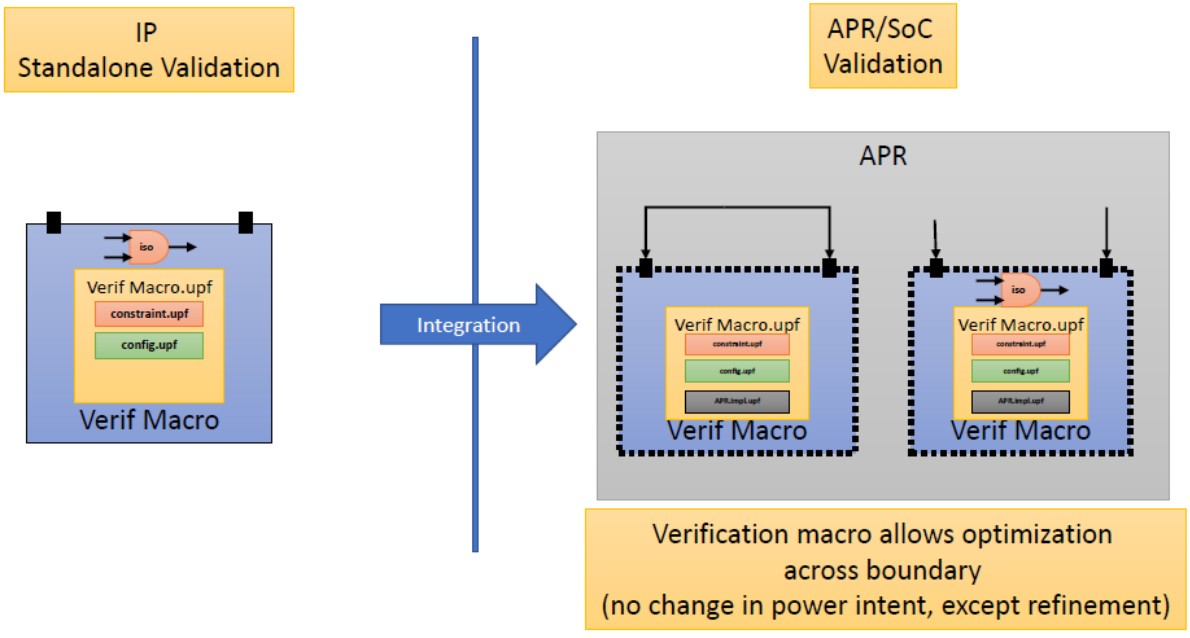

Figure 4 Bottom-up verification using PVIPs

VI. CHALLENGES AND LIMITATIONS OF THE BOTTOM-UP VERIFICATION FLOW

One of the biggest challenges with bottom-up verification flow is ensuring the integrity of PVIPs' power intent. This includes avoiding any accidental or unintentional modification of IP power intent. With the current capabilities of UPF, the user has two choices on how to capture the power intent of the PVIPs:

A. No marking for PVIPs

In this case, the UPF for PVIP is treated as any other block in the system. Users can choose to use the Successive Refinement methodology or capture the entire power intent in a single UPF. In either case, there is no mechanism to ensure the integrity of the IP power intent. In most cases, the IP integrator has to modify the PVIP UPFs to add implementation details. As a result, they have to revalidate the PVIP power intent or risk any functional issues related to power intent. The absence of any marking for PVIP limits the tool's capabilities to perform any checks to catch unintentional modifications.

B. Soft Macro marking for PVIPs

An alternative approach is marking the PVIPs as soft macros defined in UPF. However, as soft macros are intended for bottom-up implementation flows. The restrictions imposed by the language severely limit the capabilities and make them unsuitable for marking PVIPs.

Some of the soft macro's restrictions create challenges with using them to represent PVIPs. Soft macros:

- restrict optimal implementation at the APR level as they need standalone implementation.
- restrict the integrator from making SoC-specific power architecture changes to IP UPF without violating the IP-defined power intent

VII. VERIFICATION MACROS

To address the challenges and limitations of bottom-up verification flows, we are introducing a new style of macros called "Verification Macros". These macros have slightly relaxed requirements compared to soft macros, allowing users to refine the IP power intent with implementation details successfully. The verification macros still follow the self-contained UPF semantics and slightly relaxed terminal boundary semantics to ensure there is no accidental/unintentional modification from outside. The comparison of the terminal boundary semantics between soft and verification macros is shown in Table 1.

Table 1 Comparison of Terminal Boundary semantics between Soft Macro and Verification Macro

S. No.	Semantics (Ref 4.9.2.3 in IEEE 1801-2018 [1])	Soft Macro	Verification Macro
1	Driver/Receiver supply **OUTSIDE** assumes **output port (driver supplies)**, **input port (receiver supplies)**	Yes	Yes
2	Driver/Receiver supply **INSIDE** assumes **input port (driver supplies)**, **output port (receiver supplies)**	Yes	Yes
3	find_objects –transitive stops at the boundary	Yes	Yes
4	Global supply availability doesn't extend beyond boundary	Yes	No
5	Location fanout stops at the boundary	Yes	Yes
6	Cannot create any UPF objects inside macro from parent scope, except isolation/levelshifters with -location parent	Yes	Yes
7	Cannot update any UPF object inside macro from parent scope	Yes	Only allow implementation updates

A. Implementation UPF

The implementation UPF comprises UPF commands and options that provide the information needed to implement the power intent for a target technology, referred to as the "Implementation UPF" in Successive Refinement Methodology. It is expected that the commands/options will refine the original power intent without violating the original intent. Any contradictions during refinement can be caught by a tool. The implementation UPF comprises of two sets of UPF commands. The first category is the commands that create new UPF objects representing implementation-level power intent, e.g. supply ports, nets, and power switches. The second category is the commands with the -update option, which add implementation detail to an existing UPF object. The UPF LRM has built-in

semantics to catch any contradiction during the refinement process, thereby ensuring any issues are caught during the compilation stage.

The following table lists the commands and updates that are classified as representing the implementation UPF.

Table 2 UPF commands/options which represent the implementation UPF

S. No.	UPF Commands/options	Implementation UPF
1	add_power_state	No
2	add_power_state -update	-supply_expr, -legal, -illegal
3	associate_supply_set	No
4	connect_logic_net -reconnect	No
5	connect_supply_net, connect_logic_net	Yes
6	create_abstract_power_source	No (new addition to UPF 4.0)
7	create_abstract_power_source -update	-power_switch (new addition to UPF 4.0)
8	create_logic_port, create_logic_net	Yes
9	create_power_domain	No
10	create_power_domain -update	-elements, -available_supplies, -boundary_supplies, -define_func_type
11	create_power_switch	Yes
12	create_power_switch -update	-instance
13	create_supply_port, create_supply_net	Yes
14	create_supply_set	No
15	create_supply_set -update	-function
16	define_power_model	No
17	map_power_switch, map_repeater_cell, map_retention_cell	Yes
18	set_equivalent	No
19	set_isolation	Yes (would want restrictions but its hard to impose)
20	set_isolation -update	-elements, -location, -isolation_supply, -instance, -force_isolation,
21	set_level_shifter	Yes
22	set_level_shifter -update	Any refinable option as these are implementation commands
23	set_port_attributes, set_design_attributes	Yes
24	set_repeater	Yes
25	set_repeater -update	Any refinable option as these are implementation commands
26	set_retention	No (affects functionality so need to verified early)
27	set_retention -update	-instance, -retention_supply
28	set_variation	Yes
29	sim_assertion_control, sim_replay_control, sim_corruption_control	No
30	use_interface_cell	Yes

B. Enforcing implementation-only restrictions

To allow only implementation updates to UPF, a new option, -implementation, is proposed to the *load_upf* command. When this option is passed, tools can check and allow only commands that qualify as implementation UPF commands as per Table 2.

VIII. Successive Refinement of Verification Macros

The successive refinement of verification macros starts with the IP provider providing a "Constraint UPF". The IP provider also offers various "Configuration UPFs", which the IP provider has validated as a standalone block. The IP goes through exhaustive validation involving a given configuration. This validation is done at a high level without the knowledge of implementation details. During this, the IP provider will ensure IP is functioning correctly with power management, e.g.

- IP transitions into legal power states and validates the power-gating behavior
- If IP has retention, the retention registers are doing proper save/restore
- If IP has isolation, the ports are getting clamped to correct values at the right time

The IP integrator uses the constraint and configuration UPF when integrating the IP at the APR level. The integrator also adds an "Implementation UPF" that contains details specific to the implementation of the power management, which must not violate the validation already done for the IP.

IX. Benefits of Verification Macros

Treating PVIPs as Verification macros has several benefits.

1. It empowers tools to compare and highlight deviations from the original power intent, thus enabling them to catch any errors during the refinement process by allowing only *load_upf -implementation* updates during the bottom-up verification flow
2. Enables IP providers to abstract the design with respect to power management but independent from implementation details
3. Provides flexibility to SoC integrator to allow optimization of power intent implementations
4. Safely refine the power intent of the PVIP for a target technology
5. Reuse the validation done by the IP provider

X. Examples

Example 1:

A PVIP has a GATED and Always ON (AON) domain. The GATED domain is switched through an SoC implemented power switch. IP implements isolation between AON and GATED domains and validates. SoC is the implementer of the IP and needs the flexibility of picking the location of isolation coded and validated by the IP based on APR requirements.

ip.upf

```
# IP hierarchy
# ip1_top in AON domain
# |_ ip1_pgd_wrapper in PGD domain

set_design_attributes -models . -attribute {is_verification_macro TRUE}

create_supply_set ss_AON
create_supply_set ss_PGD

create_power_domain AON -elements {.}

create_power_domain PGD -elements {ip1_pgd_wrapper}

.....

## isolation strategy for ports crossing from PGD to AON.
## -location not specified by IP team
##  -location, if specified and hardcoded, cannot be overridden through
Verification macro refinement
set_isolation "o_PGD_to_AON" \
 -domain "PGD" \
 -isolation_supply_set "ss_AON" \
 -clamp_value "0" \
 -elements {
    ip1_pgd_wrapper/portA
```

```
    ip1_pgd_wrapper/portB
 } \
 -isolation_signal pwr_manager/iso_en_b \
 -isolation_sense low

....

create_pst ip_PST   -supplies "ss_AON.power ss_PGD.power ss_AON.ground"
add_pst_state ON    -state {ON  ON  ON} -pst ip_pst
add_pst_state GATED -state {ON  OFF ON} -pst ip_pst
add_pst_state OFF   -state {OFF OFF ON} -pst ip_pst
```

IP defines AON and PGD power domains. PGD is a switched domain of AON requiring isolation between them which is implemented using the *set_isolation* command. The isolation strategy has all the options required for the IP to code and validates its power intent. IP does not need a location specified as it is an implementation choice. IP provides SoC the ability to define the location based on APR needs by excluding the definition of isolation in IP isolation strategies and marking the IP as a verification macro through the *set_design_attributes -is_verification_macro* option.

ip.socimpl.upf

```
set_isolation "o_PGD_to_AON" \
 -domain "PGD" \
 -location parent \
 -update
```

SoC integrator of IP follows 2 steps to implement the location of choice.

Step 1 - ip.socimpl.upf, which refines IP defined isolation strategy with SoC's desired location. It follows successive refinement semantics to achieve the refinement with verification macro semantics, allowing isolation to be refined.

parIP1.upf

```
# SoC hierarchy
# parIP1
# |_ ip1 ip1_top

create_power_domain par_AON -elements {.}

## -implementation option is a cue to the tools that refinement happened on
IP UPF
## verification macro rules come into play and check to ensure IP power intent
## remains unchanged
load_upf ip.upf -scope ip1
load_upf ip.socimpl.upf -scope ip1 -implementation
```

SoC level parIP1.upf instantiates the original IP UPF along with the SoC refined UPF. SoC refined UPF is loaded into the same scope as the original IP UPF but with a new option *-implementation*. This new option provides a trail of changes from IP to SoC so EDA tools can check to ensure PVIP power intent remains unchanged due to SoC-driven changes.

Example 2:

PVIP delivered is placed in a shallower (less ON) power domain at SoC. The partition boundary at SoC APR block is always ON. This setup translates into a need for the output ports of PVIP to be isolated through SoC inserted isolation strategy before they get consumed.

Example 2 illustrates this case using 1 IP output port which is powered by SoC_VNN that is port mapped to SoC port which is powered by SOC_VNNAON. SOC_VNN is less ON than SOC_VNNAON. SoC inserts needed isolation from SOC_VNN to SOC_VNNAON but, due to APR constraints, pushes into IP Always ON domain.

IP.upf

```
# IP hierarchy
# ip1_top in AON domain
# | --- output port ip1_top.o_portA @ ss_AON
# |_ ip1_pgd_wrapper in PGD domain

set_design_attributes -models . -attribute {is_verification_macro TRUE}
```

```
create_supply_set ss_IP_AON
create_supply_set ss_IP_PGD

create_power_domain AON -elements {.} -primary ss_IP_AON

create_power_domain PGD -elements {ip1_pgd_wrapper} -primary ss_IP_PGD

....

create_pst   ip_PST          -supplies   "ss_IP_AON.power   ss_IP_PGD.power
ss_IP_AON.ground"
add_pst_state ON    -state {ON  ON  ON} -pst ip_pst
add_pst_state GATED -state {ON  OFF ON} -pst ip_pst
add_pst_state OFF   -state {OFF OFF ON} -pst ip_pst



set_port_attributes \
-receiver_supply "ss_IP_AON" \
-driver_supply "ss_IP_AON" \
-ports "o_portA"
```

IP.upf describes the creation of AON and PGD supply sets. PVIP expects both AON and PGD to be delivered by the SoC. It describes the relationship between AON and PGD in the power state table. IP inserted isolation, and the rest of the UPF content is not shown for brevity. Port, o_portA, is mapped to ss_IP_AON, the always ON supply from an IP standpoint.

parIP1.upf

```
# SoC hierarchy
# parIP1
# |_ ip1(ip1_top)

create_power_domain par_AON -elements {.}

create_supply_set ss_SOC_AON
create_supply_set ss_SOC_VNN
create_supply_set ss_SOC_PGD

load_upf ip.upf -scope ip1
associate_supply_set {ss_SOC_VNN ip1/ss_IP_AON}
associate_supply_set {ss_SOC_PGD ip1/ss_IP_PGD}

set_isolation "o_SOCVNN_to_SOCAON" \
 -domain "par_AON" \
 -location other \
 -elements {ip1/o_portA} \
 -applies_to_boundary lower \
 -isolation_supply ss_SOC_AON \
 -isolation_signal soc_isol_en_b \
 -isolation_sense low

.....

create_pst par1_PST   -supplies "ss_AON.power ss_VNN.power ss_AON.ground"
add_pst_state ON    -state {ON  ON  ON} -pst pa1_pst
add_pst_state GATED -state {ON  OFF ON} -pst par1_pst
add_pst_state OFF   -state {OFF OFF ON} -pst par1_pst

set_port_attributes \
-receiver_supply "ss_SOC_AON" \
-driver_supply "ss_SOC_AON" \
-ports "ip1/o_portA"
```

parIP1.upf describes the loading of IP.upf and mapping of IP supply sets AON and PGD to SOC_VNN and SOC_PGD. SoC inserts isolation crossing from SOC_VNN to SOC_AON on IP port o_portA. This isolation strategy must be implemented within the IP scope due to APR restrictions. Verification macro allows for refinement, whereas soft macros do not in situations like

1. Pushing the isolation strategy into IP scope through the use of *-applies_to_boundary lower* and location of the isolation is self at IP AON domain through the use of *-location other*.
2. Allows instance-based override of driver/receiver supplies on *set_port_attributes*. This capability comes in handy as set_port_attributes of o_portA needs an update from ss_IP_AON, which at SoC is mapped to SOC_VNN, but with SoC isolation inserted as self in IP AON domain, its attribute changes to SOC_AON.

Example 3:

Example 3 illustrates the PVIP with a GATED domain and AON models path-based isolation that inserts isolation only when there is an electrical need. PVIP choice of isolation strategy coding is to avoid redundant isolations.

IP.upf

```
set_design_attributes -models . -attribute {is_verification_macro TRUE}

create_supply_set ss_IP_AON
create_supply_set ss_IP_PGD

create_power_domain AON -elements {.}
create_power_domain PGD -elements {ip1_pgd_wrapper}

.....

## IP inserted isolation from PGD.
## Isolates all output where different supplies power source and sink
set_isolation "o_PGD_to_AON" \
 -domain “PGD” \
 -isolation_supply_set "ss_IP_AON" \
 -clamp_value "0" \
 -location self \
 -applies_to outputs \
 -source ss_IP_PGD \
 -sink ss_IP_AON \
 -diff_supply_only TRUE \
 -isolation_signal pwr_manager/iso_en_b \
 -isolation_sense low

....

create_pst   ip_PST          -supplies   "ss_IP_AON.power   ss_IP_PGD.power
ss_IP_AON.ground"
add_pst_state ON    -state {ON  ON  ON} -pst ip_pst
add_pst_state GATED -state {ON  OFF ON} -pst ip_pst
add_pst_state OFF   -state {OFF OFF ON} -pst ip_pst
```

parIP1.upf

```
# SoC hierarchy
# parIP1
# |_ ip1(ip1_top)

create_power_domain par_AON -elements {.}

create_supply_set ss_SOC_AON
create_supply_set ss_SOC_VNN
create_supply_set ss_SOC_PGD

## ip1 which is an instance of IP.upf where SoC is shorting AON and PGD
load_upf ip1.upf -scope ip1
```

```
associate_supply_set {ss_SOC_AON ip1/ss_IP_AON}
associate_supply_set {ss_SOC_AON ip1/ss_IP_PGD}



## ip2 which is an instance of same IP.upf where SoC has AON and PGD
implemented separately
load_upf ip1.upf -scope ip2
associate_supply_set {ss_SOC_AON ip2/ss_IP_AON}
associate_supply_set {ss_SOC_PGD ip2/ss_IP_PGD}

....

create_pst     ip_PST     -supplies     "ss_IP_AON.power     ss_IP_PGD.power
ss_IP_AON.ground"
add_pst_state ON    -state {ON  ON  ON} -pst ip_pst
add_pst_state GATED -state {ON  OFF ON} -pst ip_pst
add_pst_state OFF   -state {OFF OFF ON} -pst ip_pst
```

parIP1.upf illustrates IP.upf instantiated twice. SoC shorts ss_IP_AON and ss_IP_PGD for one instance (ip1) and provides separate power for the second instance (ip2).

Bottom-up verification done through a soft macro model would have resulted in redundant isolations for the ip1 instance at SoC, leading to APR congestion. The same bottom-up verification done through a verification macro semantics allows optimization at SoC APR across terminal boundaries resulting in an optimal QoR.

XI. CONCLUSION

PVIPs are becoming more and more common in current SoCs. The time spent on validation is far more than the actual design process. Reusing the validation already done for the IPs is vital to achieving more efficiency and a faster turnaround time. The current methodologies have several hurdles and limitations for correctly applying successive refinement for PVIPs. The paper proposes verification macros that allow the Successive Refinement Methodology to reuse PVIP's power intent and validation.

XII. STANDARDIZATION UPDATE

The IEEE 1801-UPF WG already approved the verification macro and its semantics. Currently, the committee is working to include the details in the next revision of the UPF 3.1 standard[1]. However, minor changes, especially concerning the naming of the verification macro marking, may be possible when the actual standard is released.

XIII. REFERENCES

[1] IEEE Std 1801-2018 IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems, 2018.