Successive Refinement of UPF Power Switches

Prabhakar Satya Ayyagari
prabhakar.s.ayyagari@intel.com
Intel

William G Crocco
william.g.crocco@intel.com
Intel

***Abstract*-The increasing complexity of soft IPs require soft IP teams to perform exhaustive validation of the IP as a standalone block, including the power management architecture. As many soft IPs have internal power gating, validating the power-switching behavior is required before delivering the IP for integration and implementation. UPF's Successive Refinement methodology enables the reuse of power intent and validation done for the IP. However, the methodology has several limitations for power switches. This forces users to use intrusive methods and edit the original UPF to introduce implementation constraints which involve modifying the simple power switches to ones that allow effective di/dt management. This paper details a new methodology to enable the Successive Refinement of power switches. The proposal is already approved by the UPF committee and will be part of the next revision of the UPF 3.1.**

I. INTRODUCTION

As power budgets continue to tighten across all semiconductor market segments, the use of low power architectures is expanding and increasing in complexity. The IEEE1801 Unified Power Format, UPF, standard defines power intent for a design to achieve the desired power–performance tradeoff. Leakage power can be reduced by implementing a gated power domain for logic in standby during a low power state. The UPF describes the implementation details of the gated domain through the use of power switches. A power switch disconnects the power or ground source from the logic that will go into standby or sleep mode. At this time, all leakage power is eliminated since no power is being provided to the devices in that domain. The power switches can be modeled in various configurations based on the power micro-architecture and di/dt mitigation requirements.

Soft IP teams are expected to deliver IP drops that are fully validated, including the power management flows. Gated domains are commonly used in soft IPs to reduce their leakage targets, but the power switch configuration implemented in the IP's UPF is not guaranteed to match the SoC's power switch requirements. The SoC's requirements are a function of power micro-architecture and process-related requirements for the di/dt mitigation.

This paper introduces the implementation concepts of a power switch in a soft IP and the current integration method supported by the UPF standard today. Limitations of the current method are also discussed, along with the restriction and costs imposed on the SoC. Finally, a proposal is made to abstract the definition at the IP level to enable power validation and allow the SoC to refine that definition to meet their power micro-architecture and di/dt mitigation needs.

II. NEED FOR POWER SWITCH REFINEMENT

Power switches are controlled through power management units. This is typically done using two configurations, one using a control signal and an acknowledge signal (to determine the following protocol, e.g., the release of isolation) or another using a control signal and a timer instead of an acknowledge signal.

When integrated into a synthesized and routed (APR) block, soft IPs implemented power switches need refinement to match micro-architecture and process-specific implementation needs for APR and power delivery. There is a significant shift within the industry to using two-stage power switches for effective di/dt management (Figure 1). Implementing power switches through daisy chains, fishbone, or other power switch placements are process and implementation choices. There is no need for soft IP teams to understand, implement or validate implementationspecific di/dt management details to complete their power management flow validation.

The successive refinement of the power switches shown in Figure 1 is not possible using the existing 1801 UPF language capabilities. Integration teams must use intrusive transformation methods, which tools cannot check. The intrusive methods include taking the original UPF verified by the IP and modifying the power switch definitions to

meet the SoC's needs. These methods are laborious, error-prone, and risk fundamentally altering the power intent that the IP validated. The proposal of using an abstracted power switch is to allow implementation teams to transform Soft IP power intent to implementation details while keeping it tool verifiable. This abstract power switch enables the soft IP to fully validate their power-gated domains without describing implementation details. If a soft IP can avoid implementation details, there is better reuse, lower IP UPF overhead and maintenance, and more flexibility for SoCs.

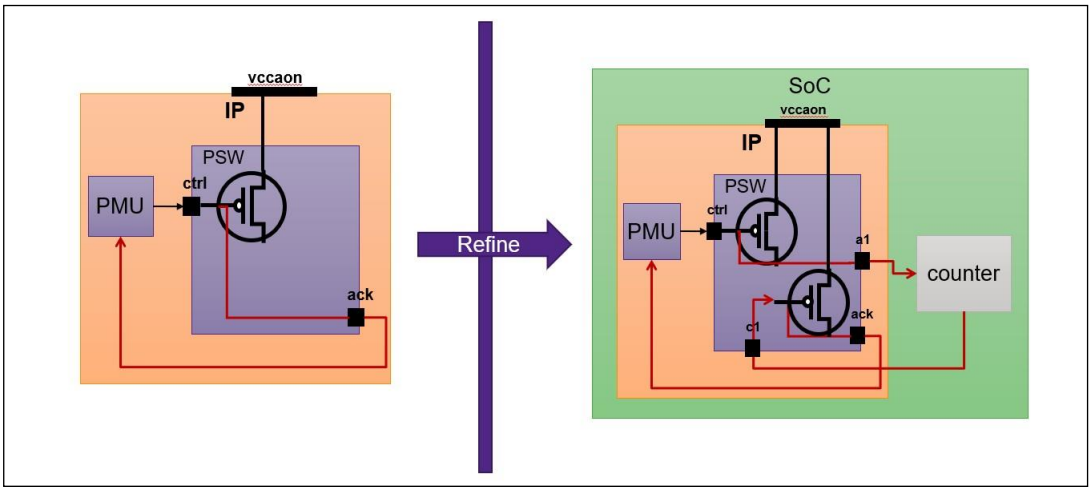

Figure 1: Sample di/dt management scheme

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.

The application of this methodology to power switches involves the following

- Capturing the power switching in an abstract form so that it can be verified at the RTL level
- The abstract power switch will be present in Configuration UPF.
- The abstraction should not limit the implementation of the power switch, as this will inhibit the reuse of the abstracted definition
- When the IP gets implemented with real power switches, it should be possible to refine and correlate the abstracted form with the actual power switch definition. This implies that any earlier validation of the switching should still hold when additional details are provided. If there is any violation, it should be flagged and caught early.

IV. CHALLENGES/LIMITATIONS WITH THE CURRENT UPF APPROACH

The current UPF LRM provides two approaches to represent the power gating behavior for verification. The first approach uses *create_power_switch*, which is the most common, and the second uses *add_power_state*. Each approach has various limitations, creating problems when refining power switches.

A. Using create_power_switch for power gating

The power switching in UPF LRM is defined using the *create_power_switch* command, which inserts switches that drive supply nets which then supply rails. Since it is based on supply nets, it requires some implementation-related choices, e.g. deciding whether the design will have a header (power) or footer (ground) switching. Hence, it is considered an implementation detail and is typically present in "Implementation UPFs" which comes late in the flow and cannot be used for early validation using the "Constraint UPF" and "Configuration UPF". A soft IP provider would like to avoid enforcing such a choice early in the flow.

The *create_power_switch* command creates more problems with the Successive Refinement Methodology for modeling the scenario described in Figure 1: Sample di/dt management scheme. It breaks the methodology's basic principle based on adding more details to the same object without contradicting the original intent. We have tried to explore the possibility of extending the UPF LRM, introducing additional refinement capabilities in the command, and

allowing adding additional ports. We hit a roadblock as adding more details could not achieve the transformation. Instead, it requires modifying the original connectivity, which contradicts the original intent. Because of these problems, the approach involving *create_power_switch* is unsuitable for the Successive Refinement Methodology.

B. Using add_power_state for power gating

An alternate approach is to capture the effect of power-switching behavior in the form of power states using the *add_power_state* command with the option *-logic_expr*. This allows users to simulate corruption of logic, using simstate CORRUPT but has the following limitations regarding the Successive Refinement Methodology.

- Power states don't allow modeling of ack logic (driving the ack port and ack_delay), which is needed for handling power controllers that depend on acknowledgment after switching
- Power states allow successive refinement by creating new states that refine the original state, e.g. ON.TURBO, ON.SLOW are refinements of ON. While this fits into the Successive Refinement Methodology, it cannot model the refinement of power switches by adding additional controls. Refer subsection "Limitations with power state refinement" for more details
- Refinement of power states has limitations when OR operators are used for definition. The OR operators are needed to capture the power-switching behavior completely

Due to these limitations, the users are forced to use intrusive transformation methods to capture power-switching behavior. These methods severely lack correlation and cannot be checked by tools – thereby losing the earlier validation performed at the IP level.

Limitations with power state refinement

If the original IP were to capture the switching behavior and the related corruption using the *add_power_state* command, it would be as follows:

```
add_power_state PD1.primary
    -state {OFF \
    -logic_expr {IP_FET_EN || PDTOP.primary == OFF}
    -simstate CORRUPT} \
    -state {ON \     -logic_expr {!IP_FET_EN && PDTOP.primary == ON}
```

The waveforms corresponding to the above *add_power_state* is shown in Figure 2.

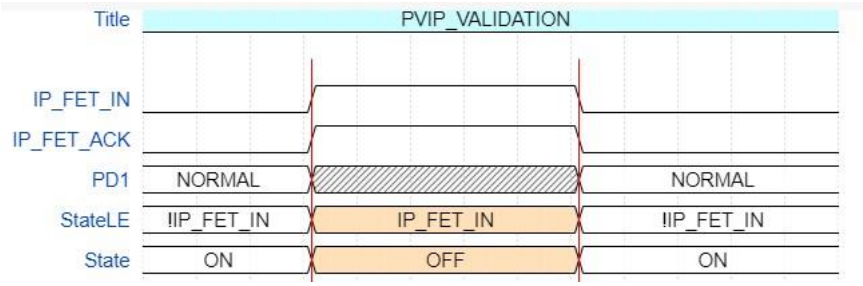

Figure 2 Waveforms during the validation of the IP

When this IP is integrated into the SoC, the *create_power_switch* commands will be added at the IP scope as an implementation update. Additionally, the *add_power_state* commands in the IP UPF must be refined/modified to match the power switching behavior. The waveform with the *create_power_switch* commands would be different as it will include the effect of additional controls and acks. This will not align with the waveforms resulting from the *add_power_state* commands from the IP UPF. Figure 3 highlights the problem.

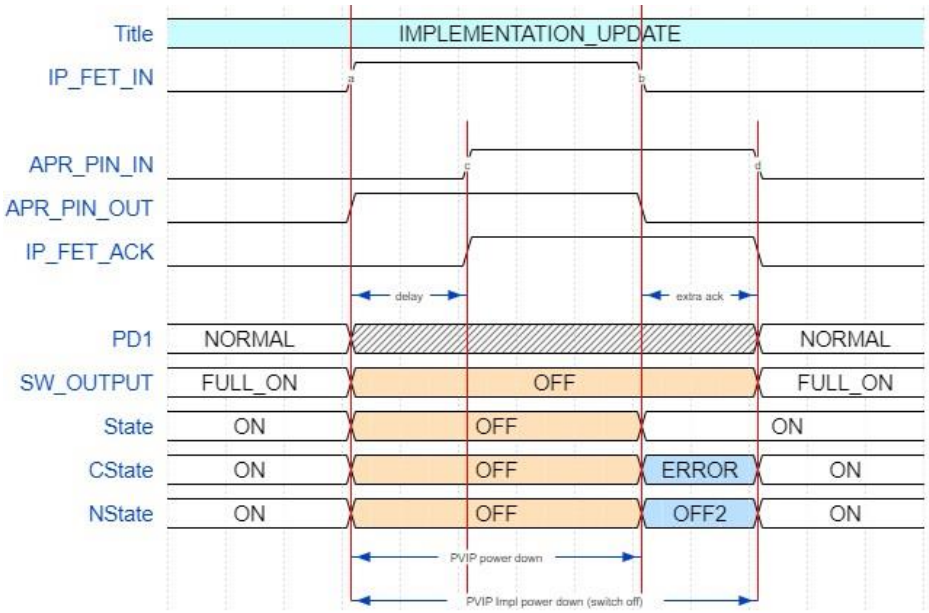

Figure 3 Waveforms based on the create_power_switch commands

To avoid the ERROR state, the *add_power_state* commands need to be re-defined to match the waveforms with the *create_power_switch* commands.

```
add_power_state PD1.primary
    -state {OFF \
    -logic_expr {IP_FET_EN || APR_PIN_IN || PDTOP.primary == OFF}
    -simstate CORRUPT} \
    -state {ON \
    -logic_expr {!IP_FET_IN && !APR_PIN_IN && PDTOP.primary == ON}
    -simstate NORMAL}
```

The current UPF standard does not permit the refinement or override of the power states with the addition of expression. Therefore, we would need to create additional states that are refinements of the fundamental power states ON/OFF.

```
add_power_state PD1.primary      -state {OFF.refined \
    -logic_expr { APR_PIN_IN }
    -simstate CORRUPT} \
    -state {ON.refined \
    -logic_expr { !APR_PIN_IN }
    -simstate NORMAL}
```

However, these refined states need to be a subset of the parent states, i.e, whenever the refined state is active the parent state also needs to be active. This holds good for the ON state, since an additional expression is ANDed , but not for the OFF state,where an additional expression needs to be ORed thereby extending the state. So, the period when ctrl is high, but ctrl_tmp is low, the “ON” state is active whereas the state “ON.refined” is not active. Also the “OFF” state is not active, therefore the current state of the supply set is ERROR.

V. ABSTRACT POWER SOURCE

To overcome the challenges, we introduce the concept of an abstracted power switch based on supply sets. Abstracting the power switching from supply nets to supply sets provides greater flexibility to the user during implementation. The abstract power switch is created using a new command *create_abstract_power_source*. The command creates an abstract power source that can be used to model not just power switches but can represent LDOs and other power sources in the future. The abstracted power switch creates user-defined supply set handles and ON and OFF states. The ON and OFF states can be referenced by power states of the primary supply of target power domains to establish supply relationships. The supply relationships can be verified separately without any implementation details. The abstracted power switch can also model the ack port driving logic at an RTL level. Finally, the abstracted power switch can be refined with real power switches created using the *create_power_switch* command.

Proposed syntax of create_abstract_power_source command:

Syntax	**create_abstract_power_source** *power_source_name* **[-output_supply_set {***supply_set_name* [*supply_set_ref*]**}]** {**-input_supply_set {***supply_set_name* [*supply_set_ref*]**}**}* {**-control_port {***port_name* [*net_name*]**}**}* {**-on_state {***state_name input_supply_set* **{***boolean_expression***}}**}* [**-off_state {***state_name* **{***boolean_expression***}}**]* [**-supply_set** *supply_set_ref*] [**-ack_port {***port_name net_name***}**]* [**-ack_delay {***port_name delay***}**]* [**-ack_polarity {***port_name* <**active_high**|**active_low**>**}**]* [**-power_switch {{***power_switch_name***}*}**] [**-update**]

create_abstract_power_source defines an abstract power source in the current scope with the name *power_source_name* that mimics the power gating behavior by controlling the ON and OFF states defined on the output supply set handle. The options *-output_supply_set* and *-input_supply_set* define local supply set handles on the power source object. These supply set handles are associated with the supply set objects defined in the option or using the *associate_supply_set* command. The *-on_state* and *-off_state* options define the conditions under which the ON and OFF states will be activated, respectively. The values on the *-output_supply_set* will be based on associating the *-input_supply_set* to the *-output_supply_set* when the ON state is active. When the OFF state becomes active, it will not affect the values of the *-output_supply_set* but only impact the simulation by triggering OFF state. The values on the functions of the supply set will depend on the final refined power switches. It shall be an error if both ON and OFF states are active at the same time. An example code of an abstract power source is shown below:

```
create_abstract_power_source PSW -input_supply_set {input PDTop.primary}
-output_supply_set {output PD1.primary}
-supply_set PDTop.primary
-control_port {en IP_FET_EN}
-ack_port {IP_FET_ACK}
-ack_polarity active_low
-on_state {PSON input {!en}}
-off_state {PSOFF en}
```

The creation of the abstract power source (as shown above) defines the ON and OFF states for the output supply set are implicitly created by mapping to the state of the abstract source. The tool will internally define the *-logic_expr* of the ON and OFF states as shown below.

```
  ## PSW.input == ON is implied when referring to PSW.output == ON. It is included
into the semantics
add_power_state PSW.output ON -logic_expr {PSW == PSON} add_power_state PSW.output
OFF -logic_expr {PSW == PSOFF}
```

The *add_power_state* commands in the IP UPF are described in terms of the state of the output of the abstract power source (as shown below), so that when it is refined to *create_power_switch* commands, the *add_power_state* commands automatically reflect the behavior of the actual power switches.

```
add_power_state PD1.primary ON -logic_expr {PSW.output == ON}
add_power_state PD1.primary OFF -logic_expr {PSW.output == OFF} -update
```

VI. SUCCESSIVE REFINEMENT OF POWER SOURCE

The abstract power switch in the IP UPF is refined during the integration into the SoC environment. The integrator will create an implementation UPF, which refines the IP UPF to include the SoC requirements. The first implementation UPF update for the IP would contain the *create_power_switch* commands for the physical implementation. The *create_logic_port* commands are used to create the additional control and ack signals. In the same UPF, a link is created between the abstract power source and the *create_power_switch* commands using the *power_switch* option to the *create_abstract_power_source* command along with *-update*. This command associates the abstract power source with the *create_power_switch* commands that will be used for physical implementation. This enables tools to check if the power intent described by the *create_power_switch* commands is different from the power intent described by the abstract power source associated with them.

The implementation updates to the IP UPF and the SoC UPF are shown below

```
IP.impl.upf ------------
create_logic_port APRPIN_IN
create_logic_port APRPIN_OUT


create_power_switch "SOCPSW_1" \
  -domain             "PD1" \
  -input_supply_port  {TVDD PDTop.primary.power} \
  -output_supply_port {VDD PD1.primary.power} \
  -control_port       [list SLEEPIN1 APRPIN_IN] \
  -control_port       [list SLEEPIN2 IP_FET_EN] \
  -ack_port           [list SLEEPOUT1 IP_FET_ACK SLEEPIN1] \
  -ack_port           [list SLEEPOUT2 APRPIN_OUT SLEEPIN2] \
  -on_state           {SW_ON TVDD "!SLEEPIN1 & !SLEEPIN2"} \
  -off_state          {SW_OFF "SLEEPIN1 | SLEEPIN2"}


create_abstract_power_source PSW
-update
-power_switch {SOCPSW_1}
```

The following update to the *add_power_state* commands is implicit based on the association of the *create_power_switch* commands to the abstract power source.

```
## Below add_power_state lines are implied.
add_power_state PSW.output ON \
  -logic_expr {PSW == PSON & SOCPSW_1 == SW_ON } -update
# Effective expr: { !IP_FET_EN & !APRPIN_IN } add_power_state PSW.output OFF \
  -logic_expr {PSW == PSOFF | SOCPSW_1 == SW_OFF } -update
# Effective expr: { IP_FET_EN | APRPIN_IN }
```

The SoC UPF makes connections to the newly created logic ports on the IP boundary

```
SoC.refinement.upf (implementation.upf)
-------------------- create_supply_set
SOC_AON

load_upf ${PATH}/ip.upf -scope ip1
associate_supply_set {SOC_AON ip1/PDTop.primary} load_upf
${PATH}/ip.impl.upf –scope ip1

connect_logic_net APRPIN_IN -ports apr/ip1/APRPIN_IN connect_logic_net
APRPIN_OUT -ports apr/ip1/APRPIN_OUT
```

VII. EXAMPLES

Abstract power switch and refinement can be used in many IP to SoC handoff scenarios. We will use three scenarios to illustrate the definition of an abstract power switch at the IP level and its refinement at SoC level.

Example 1:

IP power manager provides a single control for power-gated domain control with no acknowledge port. IP power manager implements an internal counter instead of an acknowledge port.

IP.upf

```
create_power_domain "PDTop" -elements {.} -supply "primary ss_VDD" ....
create_power_domain "PD1" -elements {mypgdwrapper}

create_abstract_power_source PSW \ -input_supply_set
{input PDTop.primary} \
-output_supply_set {output PD1.primary} \
-supply_set PDTop.primary \
-control_port {en IP_FET_EN} \
-on_state {PSON input {!en}} \
-off_state {PSOFF en}
```

IP.upf describes the abstract power source definition that captures Example 1 power intent. Supply set handles define input and output supplies. Control port defines the RTL signal from IP power manager with no acknowledge port. IP has all its power intent captured to deliver a self-sufficient, validated UPF to SoC teams.

SoC.refined.upf

```
## Design hierarchy
## apr
## -> ip_inst1
##
## Control Ports Hier Relations
## apr.ip_inst1.IP_FET_EN (PHYSICAL PORT)
## apr.APR_SoC_FET_ACK (PHYSICAL PORT) - logic port map to
apr.ip1_inst1.APR_IP_FET_ACK

## Design assumption
## IP UPF does not implement ack = a programmable counter might have been implemented
## SoC expecting ACK back to its power manager needs to comprehend IP level program
counter
create_logic_port APR_IP_FET_ACK
```

```
create_power_switch "SoCPSW_1" \
  -domain              "PD1" \
  -input_supply_port  {TVDD PDTop.primary.power} \ ## PDTop.primary.power == VDD IP
supply port
  -output_supply_port {VDD PD1.primary.power} \ ## supply net is implict, derived by
the tool
  -control_port       [list SLEEPIN1 IP_FET_EN] \
  -ack_port           [list SLEEPOUT1 APR_IP_FET_ACK SLEEPIN1] \
  -on_state           {SW_ON TVDD "!SLEEPIN1"} \
  -off_state          {SW_OFF "SLEEPIN1"}

## SoC refined switch (SoCPSW_1) is asSoCiated with IP abstracted power source
through the -power_switch option.
The -update option provides a paper trail of the changes done from IP to SoC,
allowing tools to ensure power intent did not change
create_abstract_power_source PSW \
-update \
-power_switch {SoCPSW_1}
```

SoC.refined.upf showcases the refinement of IP-provided abstract power source to a single control and single acknowledge power switch at the SoC level. SoC is shown using a logic port as acknowledge port. Example 1 illustrates a scenario where IPs power management is through counters. Example 1 shows SoC converting IP power switch with control only to have both control and ack port. SoC needs to ensure that the delay from IP managed control signal to SoC implemented acknowledge signal is greater than or equal to IP power manager counter.

.

Refinement of IP-provided abstract power source at SoC involves two important steps:

- **Refinement:** Refinement of the abstract power source to an implementation-specific power switch which is process specific. Process specification can come from switch selection and supply port association. Supply expression refinement done in SoC.refined.upf supply ports is implicitly included into add_power_state through asSoCiation.
- **Association:** SoC refined switch is associated with IP abstracted power source through the *-update* command. The *-update* provides a paper trail of the changes done from IP to SoC, allowing tools across the SoC integration and implementation spectrum to check power intent violations.

Example 2:

IP power manager provides a single control and a single ack for power-gated domain control.

IP.upf

```
create_power_domain "PDTop" -elements {.} -supply "primary ss_VDD" ....
create_power_domain "PD1" -elements {mypgdwrapper}

create_abstract_power_source PSW \ -input_supply_set
{input PDTop.primary} \
-output_supply_set {output PD1.primary} \
-supply_set PDTop.primary \
-control_port {en IP_FET_EN} \
-ack_port {IP_FET_ACK} \
-ack_polarity active_low \
-on_state {PSON input {!en}} \
-off_state {PSOFF en}
```

IP.upf describes the abstract power source definition that captures power intent of a single control from the IP power manager and a single ack back to it.

SoC.refined.upf

```
## implementation also uses single control/single ack power switch
## delays assumed to be inserted through APR flow. Not design managed

## refinement
create_power_switch "SoCPSW_1" \
  -domain              "PD1" \
  -input_supply_port  {TVDD PDTop.primary.power} \ ## PDTop.primary.power == VDD IP
supply port
  -output_supply_port {VDD PD1.primary.power} \ ## supply net is implict, derived by
the tool
  -control_port       [list SLEEPIN1 IP_FET_EN] \
  -ack_port           [list SLEEPOUT1 IP_FET_ACK SLEEPIN1] \
  -on_state           {SW_ON TVDD "!SLEEPIN1"} \
  -off_state          {SW_OFF "SLEEPIN1"}

## asSoCiation of abstract power source at IP to refined implementation power switch
at SoC
create_abstract_power_source PSW \
-update \
-power_switch {SoCPSW_1}
```

SoC.refined.upf showcases the refinement of IP single control and single acknowledge abstract power source to an SoC implementation-specific single control and single acknowledge power switch. SoC power switch adds processspecific details as described in Example 1. It is followed by the asSoCiation of the SoC refined power switch to IP abstract power source to ensure that power intent isn't violated during the SoC level refinement of the IP power source.

Example 3:

IP power manager provides a single control and a single ack for power-gated domain control. SoC implements IP power intent using a primary-secondary power switch to account for di/dt management, including implementationspecific daisy chain and fishbone power switch details.

IP.upf

```
create_power_domain "PDTop" -elements {.} -supply "primary ss_VDD" ....
create_power_domain "PD1" -elements {mypgdwrapper}

create_abstract_power_source PSW \ -input_supply_set
{input PDTop.primary} \
-output_supply_set {output PD1.primary} \
-supply_set PDTop.primary \
-control_port {en IP_FET_EN} \
-ack_port {IP_FET_ACK} \
-ack_polarity active_low \
-on_state {PSON input {!en}} \
-off_state {PSOFF en}
```

IP.upf describes the abstract power source definition that captures Example 1 power intent of a single control from the IP power manager and a single ack back to it.

SoC.refined.upf

```
## refinement
SoC
create_power_switch "SoCPSW_1" \
  -domain              "PD1" \
  -input_supply_port  {TVDD PDTop.primary.power} \ ## PDTop.primary.power == VDD IP
supply port
  -output_supply_port {VDD PD1.primary.power} \ ## supply net is implicit, derived
by the tool
  -control_port       [list SLEEPIN1 APRPIN_IN] \
  -control_port       [list SLEEPIN2 IP_FET_EN] \
  -ack_port           [list SLEEPOUT1 IP_FET_ACK SLEEPIN1] \
  -ack_port           [list SLEEPOUT2 APRPIN_OUT SLEEPIN2] \
  -on_state           {SW_ON TVDD "!SLEEPIN1 & !SLEEPIN2"} \
  -off_state          {SW_OFF "SLEEPIN1 | SLEEPIN2"}

## association of abstract power source at IP to refined implementation power switch
at SoC
create_abstract_power_source PSW \
-update \
-power_switch {SoCPSW_1}
```

SoC.refined.upf showcases the refinement of IP abstract power source single control from power manager with a single acknowledge back to an SoC di/dt controlled and process specific primary-secondary power switches. The use of two SoC-level power switches, including a counter to manage di/dt is a process and SoC-level power architecture-specific choice. The examples show the power of the abstract power source refinement proposal in handling complex refinement needs at SoC, while abstracting such details from the IP. The use of AND condition for power switch ON states and OR condition for power switch OFF states is also implicitly comprehended, allowing the SoC user to focus on the critical transformations needed while the tools heavy lift to verify that refinement did not change IP power intent.

VIII. CONCLUSION

The new command, *create_abstract_power_source*, enables capturing the power-switching behavior in an abstract form which can be refined to *create_power_switch* commands when implementation details are available. This enables tools to check if the update commands fundamentally changed the original power intent of the IP or not. Therefore it shortens the verification cycle of the SoC by not needing the re-validation of the Soft IPs (after the UPF updates) that have been validated standalone.

The IEEE 1801 UPF WG approved the command, and the changes are currently being drafted in the next revision of UPF 3.1.

IX. REFERENCES

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