Power State to PST Conversion: Simplifying static analysis and debugging of power aware designs

Madhur Bhargava, Mentor Graphics (madhur_bhargava@mentor.com)

Pankaj Gairola, Mentor Graphics (pankaj_gairola@mentor.com)

***Abstract* - The increasing use of advanced power management techniques has led to great complexities in the low power verification process. Today's chips have multiple power domains each having multiple operating power modes and dynamically changing voltage levels. Unified Power Format (UPF) provides specification of active power management for RTL designs to carry out the verification process. Accellera UPF provided commands to define the possible values of supply ports ("port states") used to deliver power to a system, together with "power state tables" (or PSTs) that defined legal combinations of port states. Verification tools and engineers relied on PST-based analysis of the possible combinations of power supply values to determine where isolation and level shifting would be required in a given design. However, it effectively required implementation of power management in detail before verification could start, tending to delay power aware verification until later in the flow than necessary or require that verification be repeated later if earlier assumptions about implementation details proved to be incorrect. Also, Accellera UPF had no way of specifying power management requirements for IP components that might be used in a system. To address these issues, IEEE Std 1801 UPF added a number of new concepts such as supply-sets and power-states that enables users to construct complex power state definitions sufficient to model various possible situations. However, this additional flexibility often results in power state definitions that are difficult to understand, difficult to debug, and even more difficult for tools to analyze. In this paper, we highlight the power aware verification challenges involved for a design having power states defined using add_power_state command. Further, we demonstrate an approach to simplify the process of static analysis and debugging for such designs. This paper also includes guidelines to define power state definitions adhering to which can help ease the verification process.**

Keywords - Power Management, Power Aware Verification, Static Analysis, Debugging Challenges in Low-Power Designs.

I. Introduction

The advent of the Unified Power Format (UPF) Standard has probably been the single biggest factor in increasing productivity when it comes to verifying the power management behavior of today's complex SoC designs. However, this increased productivity has come at the expense of increased complexity. There can be no doubt that the evolution of the UPF standard, from the original Accellera 2007 UPF 1.0 LRM, followed by the initial IEEE 1801-2009 UPF 2.0 LRM, and updated by the release of IEEE 1801-2013 UPF 2.1 LRM, has provided many new capabilities that have eased the power intent specification process as well as enabled new power management verification flows aligned with the needs of IP based SoC designs today. Unfortunately the evolution of the UPF LRM alone has not lessened the complexity of the power management verification task whatsoever. One such enhancement has been 'defining power states on a more abstract level of supply sets/power domains with the help of add_power_state command', which is an extremely flexible and powerful command that enables the user to construct very complex power state definitions sufficient to model any possible situation. However, this flexibility and power causes complexities in understanding the supply dependencies between two power domains being evaluated for static analysis. Moreover it becomes difficult to debug individual and combinations of these power states since we often end up with power state definitions that are difficult to understand, difficult to debug and even difficult for tools to analyze.

The difficulty of power management debug is reflected in the large number of EDA vendors that provide not only power aware front-end simulation and emulation capabilities but also by those focused on power management verification tools ranging from static analysis and rule based power checks, to power aware logic equivalency checking and layout tools. In this paper, we shall analyze how power states of supply sets and power domains can be used for doing static analysis to determine isolation or level shifting requirements for a given design. As power states are composed of not only supply information but also the logic information, it becomes more difficult to get the overall supply dependencies between two power domains for static analysis. By taking relevant examples and case studies, we shall demonstrate a way by which power states of supply sets and power domains can be interpreted and represented as a PST table, thereby making it simpler for static analysis and debugging of power states. We shall also highlight some of the common pitfalls that low-power designers can avoid in defining power states which otherwise can lead to complex power states definitions difficult to analyze for static analysis. Additionally, we shall highlight some of the validations for a SOC integrating multiple IP's each with power state definitions.

II. Power Intent Specification and Basic Concepts

UPF (Unified Power Format) provides mechanism to specify power intent for power management of low power designs. UPF standard is still evolving with new features, concepts and clarifications being added over the releases. Accellera UPF (UPF 1.0) included support for defining the possible values of supply ports ("port states") used to deliver power to a system, together with "power state tables" (or PSTs).

A **power state table** (PST) defines the legal combinations of port states, i.e., those combinations of port states that can exist at the same time during operation of the design. PST based analysis of the possible combinations of power supply values enables UPF-supporting tools to perform static analysis so as to determine isolation and level shifting requirements of a given design.

However, it effectively required implementation of power management in detail before verification could start, tending to delay power aware verification until later in the flow than necessary or require that verification be repeated later if earlier assumptions about implementation details proved to be incorrect. Also, Accellera UPF had no way of specifying power management requirements for IP components that might be used in a system.

To address these issues, IEEE Std 1801 UPF added a number of new features. Particular to this paper following two are of much relevance:

A **supply set** represents the power provided to a power domain for a particular use, such as the primary supply of the domain, an isolation supply, or a retention supply. Supply sets are an abstraction of connections to a power distribution network, and they can be used to model incoming power to the domain before the supply distribution network has been defined. Supply sets for a given power domain are defined along with the power domain. They can also be defined as separate objects via the create_supply_set command.

A **power state** represents a particular mode of operation of supply set or a power domain. For a supply set, a given power state indicates whether and how it is providing power to a power domain or related power management cell. For a power domain, a power state indicates the current operational mode of that domain, and as a consequence, whether or how the power domain is consuming power. Power states are defined with the add_power_state command.

III. Power States and add_power_state command

Any object consists of a set of items that characterize its functional state. For example, a supply set's state is characterized by the states of its supply set functions; a power domain's state is characterized by the states of its supply sets; an IP block's state is characterized by the states of its constituent elements (power domains and macro instances). The set of all possible combinations of values of these characteristic items is the set containing all possible functional states of the object. This set of possible value combinations defines the functional state space of the object.

A **power state** represents a subset of this set of all possible functional states of an object, or equivalently, a region within the functional state space of the object. The defining expression of a power state evaluates to True for every value combination in this subset and evaluates to False for every value combination outside this subset. Power states are defined with the add_power_state command.

The **add_power_state** command is an extremely flexible and powerful command that enables the user to construct very complex power state definitions sufficient to model any possible situation. However, that flexibility and power, if used indiscriminately, can result in power state definitions that are difficult to understand, difficult to debug, and even difficult for tools to analyze. To avoid these potential problems, it is important to understand what power states represent, how they can be defined in an orderly and methodical fashion to ensure maximum clarity, and how they can be organized to most effectively support verification and analysis by tools.

A. The add_power_state Command:

The UPF add_power_state command defines power states of a supply set or a power domain. Any number of power states can be defined for a given object. Power states of a given object are defined in terms of the states of other objects that either comprise the given object or are contained in or below the HDL scope in which the given object is defined.

Each power state is defined in terms of a supply expression, a logic expression, or both. A supply expression is used only for supply set power states; it refers to supply states of the functions of that supply set. A logic expression can be used in the definition of power states for either supply sets or power domains. It can refer to control conditions, clock frequencies, and power states of the domain's supply sets.

Each power state can be defined as either legal or illegal. If unspecified, the default is legal. A supply set power state can also specify a simstate, which defines how logic powered by this supply set will behave in simulation when in this power state. Simstate values range from NORMAL to CORRUPT, with intermediate values indicating successively more sensitivity to changes that could cause corruption.

An initial power state definition can be refined later by repeating the command with the -update option. Command refinement can extend an existing state definition - for example, to add a logic expression. It can also extend an existing state's supply expression or logic expression by ANDing the original with a new term.

```
#-----------------------------------------------------------------
# Example Power State Definitions
# Adapted from examples in IEEE Std 1801-2013, clause 6.4, pg 57
#-----------------------------------------------------------------

# Power states for the primary supply set of power domain PDA
add_power_state PDA.primary -supply \
-state {ON -simstate NORMAL \
-logic_expr {SW_ON} \
-supply_expr { power == {FULL_ON 0.8} && \
ground == {FULL_ON 0.0} } } \
-state {OFF -simstate CORRUPT \
-logic_expr {!SW_ON} -supply_expr { power == OFF || \
ground == OFF } }

# Another power state for the primary supply set of power domain PDA
add_power_state PDA.primary -supply -update \
-state {SLOW -simstate CORRUPT_STATE_ON_CHANGE \
-logic_expr {SW_ON && interval(clk posedge negedge)>= 100ns} \
-supply_expr { power == {FULL_ON 0.8} && \
ground == {FULL_ON 0.0} } }

# Declaring that there are no more legal power states of PDA.primary
add_power_state PDA.primary -supply -update -complete

# Power states of power domain PDA based on its primary supply and a control input
add_power_state PDA -domain \
-state {RUN -logic_expr { primary == ON && !sleep } } \
-state {SLEEP -logic_expr { primary == ON && sleep } } \
-state {SHUTDOWN -logic_expr { primary == OFF } }

# Power states of power domain PDTOP based on the power states of domains PDA, PDB
add_power_state PDTOP -domain \
-state {S1 -logic_expr { PDA == RUN && PDB == RUN } } \
-state {S2 -logic_expr { PDA == SLEEP || PDB == SLEEP } } \
-state {S3 -logic_expr { PDA != RUN && PDB != SHUTDOWN } }
```

B. The Power of add_power_state:

The add_power_state command is very powerful in part because the supply and logic expressions used to define power states allow for general Boolean expressions. As the above example code illustrates, these expressions include support for the following special subexpression forms:

- interval (signal name [, edge1 [, edge2]])
    - for detecting clock frequencies
- supply == {supply net state [voltage1 [voltage2]]}
    - for detecting a supply port/net or supply set function's value
- supply set == power state
    - for detecting the state of a supply set that affects another object's power state
- power domain == power state
    - for detecting the state of a power domain that affects another object's power state

Since power states are defined with Boolean expressions, they are essentially predicates. When a given power state's defining expressions are True, that power state is active. As a consequence of this definitional approach, more than one power state can be active at a given time. This allows definition of non-mutually-exclusive power states as well as mutually-exclusive power states.

C. Power State Refinement:

Refinement of a power state amounts to subsetting. Refining a given power state involves definition of a new power state characterized by a more restricted subset of the functional states of the given power state. A refined power state is therefore always contained within the functional state space of the original, more abstract power state.

Refinement (i.e., restriction) is accomplished by imposing additional requirements that must be satisfied by the set of characteristic item values that define the more abstract power state. This amounts to extending the defining expression of the more abstract power state with another condition.

Refinements can be categorized in two categories:

- In-place Refinement: Use of –update to refine power-states amounts to "refinement in place", in that the original definition is modified in the process.
- Refinement by derivation: This approach involves defining a new power state (with a new name), based on the original power state. Such refinement can be done any number of times without modifying the original power state definition. Each refinement produces a new power state that is non-mutually-exclusive with the original abstract state. This approach preserves the original power state definition and thus avoids unexpected semantic changes in other commands that refer to that original power state.

IV. Power Aware Verification: Static Analysis

In this section we shall discuss basic steps performed during static-analysis to identify level-shifter and isolation requirements of the design, using power intent specified in UPF.

A. Static analysis of the design for Level-shifter requirment:

Level shifters are normally required for the signals crossing the power domain boundaries when source and sink supplies operate at different voltage levels when they are on. Following are the steps performed by the verification tools to determine the level shifters placement requirements:

1. Identify the signals that are crossing the voltage island boundary. This can be achieved by traversing over the interface of the elements that belong to the power domain.
2. Detect the operating voltages of the two islands. Using the information from the power state table, the tool can easily figure out the operating voltage of a particular domain. Depending on whether there is a difference in the operating voltage, then the particular signal is a potential candidate for a level shifter.
3. Check if there is a level shifting strategy specified in the UPF through 'set_level_shifter' command.
4. Depending on whether 'a level shifter is required or not' and 'a corresponding level shifting strategy is specified or not' we can detect 'a valid', 'a missing' or a 'redundant' level-shifter. Further by comparing properties of specified and required level-shifter we can deduce 'incorrect' level-shifter.

B. Static analysis of the design for Isolation requirment:

Isolation cells are clamps that drive a particular value on the signal when the isolation-enable control is triggered. They are required to mask the floating values on the inputs that are driven by signals from the domain that is switched off. Following are the steps performed by the verifications tools to determine isolation cell placement requirements:

1. Identify the signals that are crossing the power domain boundary similar to what is specified for level shifters.
2. Determine whether the primary supply could be switched or not. This is done by tracing back the primary power and ground pins until they terminate at the port of a switch or a port of the domain boundary. If they terminate at the port of a domain boundary, then the switch is again retraced back until they end at a supply pad. Since a supply pad is considered to be always on, a primary power and ground supply of a domain driven by a supply pad will be a possible candidate for an always-on domain, whereas the one terminating

at the port of a switch will be a switched supply. Also, supplies that are switched can be matched with entries in a power state table to figure out whether they are switched at the same time or not.

3. Check if there is an isolation strategy specified for the signal or not through the set_isolation command.
4. Depending on whether 'an isolation is required or not' and 'a corresponding isolation strategy is specified or not' we can detect 'a valid', 'a missing' or a 'redundant' isolation.

C. Static Analysis of the design with power states:

As mentioned in earlier sections, IEEE Std 1801 UPF empowers users to specify power states for supply-sets and power-domains that enables the user to construct complex power state definitions sufficient to model various possible situation. However, with this additional flexibility UPF supporting tools have to do the analysis of possible combinations of power supply values and thus heavily rely on the power state definitions during 'static analysis phase'. The additional challenges thus posed for the static analysis of the design are discussed in detail in next section.

V. Challenges in static analysis & debugging of design with Power States

Any low power IP may operate in different power modes. These power modes are typically represented in the form of power state information in the power intent specification files. The knowledge of power states can help the IP integrator check the integration doesn't contradict IP's power modes. A power state on a supply set can also drive the simulation as it can also specify a simstate, which defines how logic powered by this supply set will behave in simulation when in this power state. These simstate values range from NORMAL to CORRUPT. In order to determine where isolation and level shifting would be required in the design and to check that isolation and level shifting is actually inserted by the UPF power intent specification where it is required, UPF supporting tools have to do the analysis of possible combinations of power supply values and thus heavily rely on the power state definitions. One of another major debug tasks is verification of the designs operational power states. This requires verifying that each defined power state of every power domain has been covered and functioning properly. It also requires verification of all power state combinations across all domains that comprise each operational power state.

Accellera UPF 1.0 standard supported the creation of power states using the three Power State Table (PST) commands, `add_port_state`, `create_pst`, and `add_pst_state`, however there were several limitations that were not addressed in the standard. The IEEE UPF 2.0 standard addressed these limitations by providing the `add_power_state` and `describe_state_transition` commands. Not only does `add_power_state` support bias states, hierarchical power state creation, and an incremental update capability, it also allows any named power state to be declared as legal or illegal. Likewise, the `describe_state_transition` command allows any transition between two power states to be declared legal or illegal.

For a low-power design operating in different power modes, debugging & analysis of power states for static verification of design are the two very important tasks. With the usage of PSTs in UPF 1.0; both the debugging and static analysis of design were relatively simpler tasks as the power state definitions were defined on supply nets/ports, and also these definitions were represented in tabular manner which was easy to interpret. Defining power states on a more abstract level of supply sets/power domains with the help of add_power_state command is an extremely flexible and powerful command that enables the user to construct very complex power state definitions sufficient to model any possible situation. However, that flexibility often result in power state definitions that are difficult to understand, difficult to debug, and even difficult for tools to analyze.

A. Static Analysis of the design with power states

- Incremental refinement of power states: One of the methodologies followed in low-power verification is to simulate and verify the design at various states of UPF implementation. Low-power design can be statically verified for isolation requirements even if the implementation UPF containing the definitions for supply nets/ports is not present. This verification was not possible with the help of PSTs as it required the actual supply nets/ports. However the analysis for isolation requirements requires analysis of simstates of driver/receiver logic. This is further explained with help of a case study in the below section [VIII].
- Getting the inter-dependencies of supply states: The power state of a system depends on the power states of its constituents IPs which are then sub-dependent on states of their supply sets. The power

states of these supply sets are further dependent on state and voltage values of supply nets and ports. In PSTs based analysis, getting these dependencies was fairly straightforward as everything was available in tabular format. However with `add_power_state` command which allows usage of various Boolean operators, getting the dependencies of supplies of two power domains is a tedious process. This problem gets even more complex, when the power state definitions for driver/receiver logic do not have direct/indirect reference of each other, however may have a common signal in power states definitions.

B. Debugging challenge – Why this power state reached?

Power state definitions on supply sets can control the behavior of logic controlled by this supply set. It is even more important to know how and when a system reaches an erroneous state. Low-power simulation vendors issue run-time error messages to indicate that an illegal power state is reached or any illegal power state transition occurs. However as the power state definitions are quite complex, it is very difficult for a user to answer the question "why this power state reached"?

UPF Code

```
add_power_state PD_CPU_SS -state ON4 { -logic_expr { PD_ALU1_SS == ON5 ||  PD_ALU2_SS == ON5 } -
simstate CORRUPT -illegal}
add_power_state PD_ALU1_SS -state ON5 { -logic_expr { !pwr_alu1 } -simstate CORRUPT }
add_power_state PD_ALU2_SS -state ON5 { -logic_expr { !pwr_alu2 } -simstate CORRUPT }
```

Simulation message

```
    # ** Error: (vsim-8933) MSPA_UPF_ILLEGAL_STATE_REACHED: Time: 129 ns, Supply set 'PD_CPU_SS'
reached an illegal power state 'ON4'. # File: src/parser_test22/demo.upf, Line: 73, Power
state:ON4
```

The simulation message denotes that the system PD_CPU_SS reached an illegal state. For a user, it is important to know the reason as to why the system reached illegal state. One of the ways would be to identify all the control signals and supply nets/ports used to construct this power state and check for their values. However as the power state definition here involved the logical operator "||", it is slightly difficult for a user to do this root cause analysis to figure the particular culprit control/supply signal.

VI. Power State to PST conversion

Every object, for which power states has been defined using UPF 2.0 command add_power_state, can be represented in the form of a power state table (PST). The power objects (power domain, supply set, supply nets, control signals) are represented as column objects in the converted power state table. The power states defined for this power object (supply set or power domain) are represented as rows objects of converted power state table. The power/supply sets state of the power objects are the entry values in the power state table.

In UPF 2.0, power state definitions using `add_power_state` commands are hierarchical in nature. That is, the power state of a system is dependent on power state of its constituent IPs (represented as power domains). The power states on power domains are dependent on power state of its supply sets which are further dependent on state of supply functions. As a result, it can be seen that every power state can be represented as combination of values of supply ports & control signals. As power state for a particular object can be represented in a power state table, where each of this combination can be represented as a row of a power state table.

The power state table representation of power states of a power object can be done at any stage of the design. In the incremental refinement of power states, the PST conversion can be done at each step of refinement. Static checking by the verification tools can be performed at any step.

A. Basic properties of this approach of power state to PST conversion:

- One power state table is constructed for one power object (supply set/power domain)
- At any point of time, the system will be in a defined state. That is at any point of time, one of the rows of this power state table will evaluate to be true.
- All the other power objects (Power Domain, Supply Sets, Control Ports/Nets, Supply Ports/Nets) referred in the power state definitions are represented as column objects in the converted power state table.
- One additional column of simState is added to the table, which represent the simstate of supply sets when the power object for which PST is created is a supply set. In case of power domain, this simState represent the

simstate of primary supply set of power domain for which PST is created. This simState helps in determining whether isolation is required during the static analysis of the low-power design.

- A power state of a power object is represented as one or more rows in the converted power state table.
- The use of Boolean operator's is limited to "&&" and "||". The "&&" operator effectively means that either the values of rows will get refined or new column's will be added to the table. The "||" operator effectively means that splitting of rows will take place, and new rows will be added in the table.
- Don't cares may get added in the cell values. This represent that the power object can be in a defined particular state even if one of the dependent power object is in unknown (but defined) state.
- This approach is in accordance with successive refinement methodology where the power state table will also get refined based on the incremental updates in power state definitions.
- As described in the case studies, consistency checks can be done at the time of conversion to catch issues in the power state definitions.
- The converted power state table can be analyzed at any point of time by the verification tools to do the static analysis for isolation/level shifting requirements. For isolation requirements the converted PSTs can be analyzed even before implemented UPF is available. This facilitates early verification of design before the supply network gets implemented.

B. Understanding with example: Power State to PST conversion

The example follows hierarchical top down approach of building up UPF. Consider the figure 1; the power state of top level design unit PDTOP is dependent on power state definitions of its constituents IPs PDA and PDB. The power states of PDA and PDB are dependent on the power state of their respective primary supply sets and control signals.

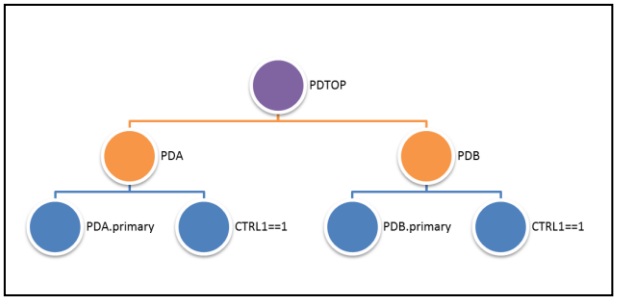


Figure 1

UPF Code

```
add_power_state PD_TOP -state TOP1 \
   { -logic_expr { PDA == A1 && PDB == B1 &&
PD_TOP.primary == TOPSS1} }
add_power_state PD_TOP.primary -state TOPSS1 \
   { -logic_expr { TOPPWR } -simstate NORMAL }
add_power_state PDA -state A1 \
   { -logic_expr { PDA.primary == ASS1 && CTRL } }
add_power_state PDB -state B1 \
   { -logic_expr { PDB.primary == BSS1 && CTRL } }
add_power_state PDA.primary -state ASS1 \
   { -simstate CORRUPT }
add_power_state PDB.primary -state BSS1 \
   { -simstate NORMAL }
```

Even at this stage when implementation UPF is not present, the power state tabular representation can still be done. Verification tool can perform static analysis to determine the isolation requirements.

Converted PSTs

PDA.primary			PDB.primary	
PDA.primary	PDA.primary.SimState		PDB.primary	PDB.primary.SimState
ASS1	CORRUPT		BSS1	NORMAL

PDA			
PDA	PDA.primary	CTRL	PDA.primary.SimState
A1	ASS1	1	CORRUPT

PDB			
PDB	PDB.primary	CTRL	PDB.primary.SimState
B1	BSS1	1	NORMAL

PDTOP										
PDTOP	PDTOP .primary	TOPPWR	PDTOP .primary .simstate	PDA	PDA .primary	CTRL	PDA .primary .SimState	PDB	PDB .primary	PDB .primary .simstate
TOP1	TOPSS1	1	NORMAL	A1	ASS1	1	CORRUPT	B1	BSS1	NORMAL

Once we have a PST based representation, verification tool can perform consistency error check to determine if the power state definitions are correct. In this case, as the CTRL signal is common in both the objects, it is added as one column. Tool checks if the value of CTRL is same for both CTRL (PDA) and CTRL (PDB) as the these objects points to same cell for TOP1 state.

In successive refinement methodology, with implementation UPF the power state definitions and corresponding power state table will get refined. New columns of supply functions will get added.

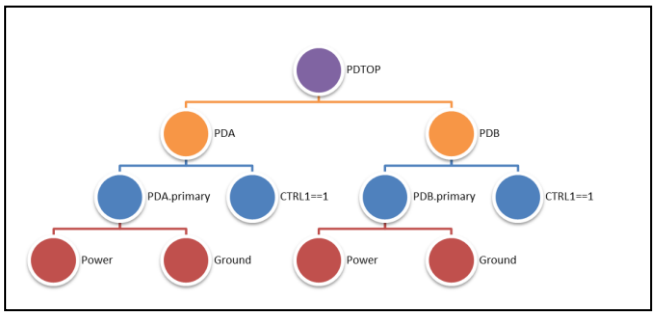


Figure 2

UPF Code

```
add_power_state PDA.primary -state ASS1 \
   -update {-supply_expr \
   (POWER == OFF) }
add_power_state PDB.primary -state BSS1 \
   -update {-supply_expr \
   (POWER == FULL_ON, 1.0) }
```

- Two new power objects `PDA.primary.power` and `PDB.primary.power` get added as column objects to to the converted PST.
- The rows are updated to capture the values of these newly added objects.

PDTOP

PDTOP	PDA	PDA .primary	CTRL	PDA .primary .SimState	PDB	PDB .primary	PDB .primary .simstate	PDA .primary .power	PDB .primary .power	…
TOP1	A1	ASS1	1	CORRUPT	B1	BSS1	NORMAL	OFF	FULL_ON, 1.0	…

Use of "||" operator can result in don't care in cell values.

UPF Code

```
add_power_state PDA -state A { -logic_expr { pwrctrl1 || pwrctrl2} }
```

PDA

PDA	pwrctrl1	pwrctrl2
A__1	*	1
A__2	1	*

VII. Case Studies

To better understand the advantages of conversion of power state to PSTs, this paper includes a few case studies.

A. Complex interaction between dependent power states

Consider the following design example of a SOC having multiple power domains; the power states of top level IPs are dependent on power state of low level IPs. Power state dependencies are as follows:

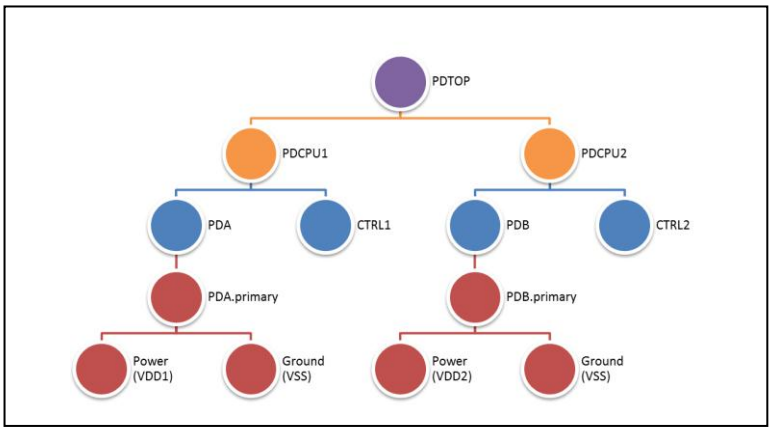


Figure 3

UPF Code

```
add_power_state PDTOP -state TOP1 {-logic_expr {PDCPU1 == C1 && PDCPU2 == C2}}
add_power_state PDTOP -state TOP2 {-logic_expr {(PDCPU1 == C1 && PDCPU2 == C3) || CTRL0} -
illegal}
add_power_state PDCPU1 -state C1 {-logic_expr {PDA == A1 && CTRL1 == 1} }
add_power_state PDCPU2 -state C2 {-logic_expr {PDB == B1 && CTRL2 == 1} }
add_power_state PDCPU2 -state C3 {-logic_expr {PDB == B1 && CTRL2 == 0} }
add_power_state PDA -state A1 {-logic_expr {PDA.primary == P1} }
add_power_state PDB -state B1 {-logic_expr {PDB.primary == P2} }
add_power_state PDA.primary -state P1 \
   {-supply_expr {(power == OFF) && (ground == FULL_ON)} -simstate CORRUPT }
add_power_state PDB.primary -state P2 \
   {-supply_expr {(power == FULL_ON, 1) && (ground == FULL_ON)} -simstate NORMAL }
```

In low-power intent specification, with the help of add_power_state commands, it is easy for the designer to specify the complex power intent. Even though the add_power_state commands only uses "&&" Boolean operator, however for the static analysis of isolation/level shifting requirement, it is still difficult for the verification tools and the verification engineer to figure out the possible combinations of supply states.

In the above example, relationship needs to be identified between PDA.primary (source) and PDB.primary (sink). With the approach of verification tools internally converting these complex power states definitions to PSTs, it will be easy to clearly identify the supply dependencies.

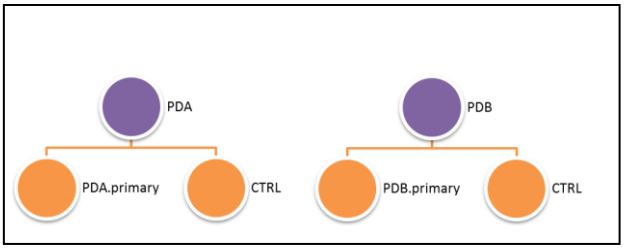
Converted PSTs (showing only relevant columns)

PDTOP	PDCPU1	PDCPU2	PDA	PDA .primary	PDB	PDB .primary	PDA .primary .simstate	PDB .primary .simstate	CTRL1	CTRL2	CTRL0
TOP1	C1	C2	A1	P1	B1	P2	CORRUPT	NORMAL	1	1	*
TOP2__1 (Illegal)	*	*	*	*	*	*	*	*	*	*	* (0)
TOP2__2 (Illegal)	C1	C3	A1	P1	B1	P2	CORRUPT	NORMAL	1	0	1

For Iso analysis, it can be easily identified from Row1 that PDA.primary can be in CORRUPT state when PDB.primary is in NORMAL state. When PDTOP is in TOP2 state, the system is in illegal state. Verification tool can highlight the corresponding row in the converted PST to show the active illegal state. From the Row3, it can be identified that CTRL0 was "1" which was the reason the system entered illegal state.

B. Identify the supply dependencies between mutually exclusive power states

In the following design scenario, the power states of PDA are dependent on its primary supply set and a control signal CTRL. The power states of PDB are dependent on its primary supply set and same control signal CTRL.



UPF Code

```
add_power_state PDA -state A1 \
  {-logic_expr (PDA.primary == AP1 && CTRL == 1) }
add_power_state PDA -state A2 \
  {-logic_expr (PDA.primary == AP2 && CTRL == 0) }
add_power_state PDB -state B1 \
  {-logic_expr (PDB.primary == BP1 && CTRL == 1) }
add_power_state PDA -state B2 \
  {-logic_expr (PDB.primary == BP2 && CTRL == 0) }
```

Figure 4

PDA	PDA.primary	CTRL
A1	AP1 (NORMAL)	1
A2	AP2 (CORRUPT)	0

PDB	PDB.primary	CTRL
B1	BP1 (NORMAL)	1
B2	BP2 (CORRUPT)	0

As it can be seen there is no direct dependency between PDA and PDB, however since the control signal is common in the power state definitions, verification tools can still identify the relation between PDA and PDB.

If we do not take consider CTRL in to account, then AP2 && BP1 is a valid combination which would mean that isolation is required. However on composing the two PSTs, we can figure that CTRL is different in two cases making the combination invalid.

PDA	PDA.primary	CTRL	PDB	PDB.primary
A1	AP1 (NORMAL)	1	B1	BP1 (NORMAL)
A2	AP2 (CORRUPT)	0	B2	BP2 (CORRUPT)

For a user, the task debugging of such static check would have been very difficult looking at these power states individually. However by this converted PST, the debugging is a much simpler task now.

C. Consistency / Error Checks in power state definitions

Consider the following example of refinement of power states.

UPF Code

```
#-----------------------------------------------------------------
# Power state "refinement in place"
#-----------------------------------------------------------------
add_power_state PDA -domain -state {A1 -logic_expr {C1} }

add_power_state PDTOP -domain -state {TOP1 -logic_expr {PDA == A1 && C2} }
# The above is equivalent to
# add_power_state PDTOP -domain -state {TOP1 -logic_expr {C1 && C2} }
#-----------------------------------------------------------------
```

PDTOP	PDA	C1	C2
TOP1	A1	1	1

```
#-----------------------------------------------------------------
add_power_state PDA –domain –update -state {A1 –logic_expr {!C2} }
# The above is equivalent to
# add_power_state PDA –domain –update -state {A1 –logic_expr {C1 && !C2} }

# And it causes a ripple effect on PDTOP
# add_power_state PDTOP –domain \
# -state {TOP1 –logic_expr {C1 && C2 && !C2} } ; # <= contradiction
#-----------------------------------------------------------------
```

The last add_power_state command cases the contradiction and is error case. Such erroneous scenarios can be easily identified and presented to user with the approach of converting power states to PSTs. Here the row1 needs to be updated with the new value of C2. However as the new value is not inaccordance with the previous value, tool will give out error message.

PDTOP	PDA	C1	C2
TOP1	A1	1	1 (old value) 0 (new value) -- error

VIII. Guidelines for modeling power states with add_power_state

UPF command add_power_state which is used to define power states on supply sets and power domains is an extremely flexible and powerful command. It allows usage of boolean operators in supply and logic expression to define the relationship between two objects. Using random Boolean operators like xor, xnor in power state definitions can result in power state definitions that are difficult to understand, difficult to debug, and even difficult for tools to analyze. Below are some of the guidelines which should be followed which will help the verification tools to convert power state definitions to PSTs thereby allowing static analysis and debugging of power state definitions. For a more comprehensive list of guidelines, please refer to [3].

- -logic_expr in power state definition of power domains should refer to control signals, power states of supply sets of this power domain and the power states of its descendants power domains.
- Avoid use of -supply_expr in power state definitions of power domains.
- -logic_expr in power state definition of supply sets should refer to control signals
- -supply_expr in power state definition of supply sets should refer to its supply functions
- In supply_expr and –logic_expr, try limiting the use of Boolean operators to "&&" and ==". Avoid use of "||" operator in –logic_expr as it can introduce don't care in power state table. Also avoid use of "!=" in -supply_expr. This brings uncertainity in the voltage and sate values of supply functions.

IX. Conclusion

To specify the power modes, power states are now defined on supply sets & power domains using UPF `add_power_state` command. As this UPF command provides lot of flexibility and power, it often results in potentially complex and difficult to understand power intent specifications. With PST's, the static analysis and debugging of design was a fairly straight forward process. However the static analysis & debugging is not intuitive with power state definitions involving supply and logic nets. In this paper, we first highlighted the power aware verification challenges involved for a design having power states defined using add_power_state command. Additionally, we demonstrated how the process of static analysis can be eased up if power states are converted and interpreted as PSTs. The paper also includes guidelines to define power state definitions following which can help ease the verification process.

X. References

[1] IEEE Std 1801™-2013 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 29 May 2013

[2] Static and Formal Verification of Power Aware Designs at the RTL Using UPF (Rudra Mukherjee, Amit Srivastava and Stephen Bailey), DVCon 2008

[3] Unleashing the Full Power of UPF Power States (Erich Marschner, John Biggs), DVCon 2015

[4] Debug Challenges in Low-Power Design and Verification (Durgesh Prasad, Madhur Bhargava, Jitesh Bansal, Chuck Seeley), DVCon 2015