Let’s DisCOVER Power States

Pankaj Kumar Dwivedi
(pankajk_dwivedi@mentor.com)
Amit Srivastava
(amit_srivastava@mentor.com)
Veeresh Vikram Singh
(veeresh_singh@mentor.com)
Mentor Graphics

***Abstract*- Power states are a key aspect of today’s low power designs and play significant role in low power verification and implementation flows. They capture the intent about the operating modes of the low power design and hence have a huge impact on the functionality. The creation of power states starts as early as the system design phase and persists through all of the implementation phases. So it becomes essential to verify the occurrence of various power states and their transitions to ensure proper operation of low power designs. A typical low power IP operates in several modes and when an SoC comprising of several of such IP is verified it becomes critical to ensure proper coverage of those states. Because the power states are captured in UPF in an abstract manner, it becomes a challenge to capture the coverage metric of those states and their transitions. In this paper, we demonstrate a methodology and flow to enable the coverage collection of power states that is generic and customizable. The paper will also highlight the importance of various metrics associated with power state coverage and the challenges faced in modelling them.**

I. Introduction

The widespread effect of power management on design functionality demands power aware verification to avoid any functional bugs. Power aware verification has become a critical part of every design and verification flow. In order to ensure that the verification process is complete, it is important to adopt a coverage-driven methodology for verification. It provides a powerful means to deal with increasing design complexity and constrained schedules. An important part of coverage-driven methodology is defining a set of metrics to be tracked during the verification process. The reliability of coverage-driven verification depends on the comprehensiveness of these metrics. Traditional metrics correspond to code and functional coverage. However, these metrics are not sufficient to capture metrics related to power management. The main reason is that power management is specified as a UPF file which contains different commands and semantics. However, for comprehensive verification it is important to define a set of coverage metrics related to power management and any associated objects. The abstract nature of these concepts makes it difficult to devise a set of metrics and use traditional approaches to achieve coverage collection.

Power States form the basis of power management. Usually, the design captures the operation in terms of power states of the design. Hence, capturing coverage for metrics related to power states play a critical role in low power verification. It is important to define metrics like state and transition coverage for the power states – which are defined in a UPF file. In this paper, we will demonstrate a coverage based methodology using standard approaches for power states.

II. Power States In UPF

Power states are intended to capture the operating modes of a low power design. It is an abstract representation of the voltage and current characteristics of the supply that is powering the portion of the design. Along with the supply information it also captures the operating modes of the elements which are present in that region. IEEE Std 1801™-2013 Unified Power Format (UPF)[1] provides a mechanism to capture the information about power states and its semantics using UPF commands. It allows attributing power states on various UPF objects, like supply ports, supply sets, power domains and system/subsystem.

A. Power states of supply nets and ports

UPF allows users to describe the supply network of the design in terms of supply nets/ports and power switches. The supply port/net carry two pieces of information: a supply state and a voltage value, which together constitute the power state of the supply net or port. UPF also provides an HDL representation of these objects in the LRM in the form of VHDL/SystemVerilog packages. In that representation, the HDL type is defined as a record/structure containing two fields corresponding to state and voltage. The state of a supply type object is an enumerated type containing the following values: FULL_ON, OFF, PARTIAL_ON, or UNDETERMINED. The voltage information is captured as integer in micro Volts and is relevant only when the state is FULL_ON or PARTIAL_ON.

B. *Power states of supply sets*

A Supply set in UPF is a collection of supply nets that together define a complete power supply. The supply set is composed of two or more supply nets. These supply nets are typically referred to as functions of the supply set. The power state for a supply set is typically captured in terms of the intended states of the supply set functions. The supply sets also carry information about the expected operation of the region to which they supply power. This operation is captured in terms of simstates that define the simulation behavior, such as CORRUPT, CORRUPT_ON_ACTIVITY, etc.

C. *Power states of power domains*

A power domain is a collection of HDL module instances and/or library cells that are treated as a group for power management purposes. The power state of a power domain is determined by the state of each of the supply sets associated with the domain and any dependent supply objects.

D. *Power States of System/subsystem*

UPF also allows capturing the state information of a system which is composed of various IPs. This captures various operating modes for a given system which is dependent on the operating modes of constituting IPs. In UPF, users can achieve this by creating a representative power domain for the system and adding states on that power domain in terms of states of the IP power domains and other supply sets.

UPF defines the add_power_state command to capture information about power states. The syntax of the command is shown below:

```
  add_power_state object_name
  {-state state_name {[-supply_expr {boolean_function}] [-logic_expr
{boolean_function}]
  [-simstate simstate][-legal | -illegal] [-update]}}
  [-simstate simstate][-legal | -illegal]
  [-update]
```

Using the add_power_state command, the user can add states on supply sets and power domains. It accepts a string parameter that adds the state of the given name on the object. It allows the user to capture various dependencies and predicates in the form of –logic_expr and –supply_expr options. These options accept a Boolean expression which captures the relevant logic conditions and other state dependencies. This command also allows the user to capture the simstate information and also specify the legality information related to the power state. One of the advantages of the add_power_state command is that the user can specify hierarchical dependencies by directly referring to power states of other objects. This minimizes the verbosity of the expression.

III. Why coverage of Power States?

In order to achieve comprehensive verification related to power states, a verification engineer is interested in a number of questions, such as:

1. All the desired power states reached or not?
2. All desired power state transitions reached or not?
3. Any illegal power state reached?
4. Any illegal power state transition occurred?

These kinds of questions are easily addressed by coverage-driven verification. But, it becomes a challenge to capture the coverage information of power states due to following two reasons:

1. Power states are written in an abstract manner in UPF, and
2. There is no pre-defined coverage metric to capture power states and their transitions.

IV. Coverage Metrics for Power States

In order to address these challenges, we must first define certain coverage metrics to capture power state coverage information. We can define three power state coverage metrics, as follows:

A. *State Coverage*

State coverage is the basic metric which ensures that all the power states are exercised during verification. It is analogous to traditional FSM coverage but is asynchronous in nature. Another important characteristic of the power states is the additional information they provide about the behavior of the system during each state – also referred to as simstates (CORRUPT, NORMAL, etc.). These power states directly impact critical low power functionalities such as simulation behavior (CORRUPTION, NORMAL etc.), isolation, level shifting, retention, etc. Hence, one has to make sure that all the states are hit. This will ensure that all the corresponding low power functionalities have been verified and their impact on actual design objects have been observed as tested.

B. Transitions, Described State Transitions and Multi State Transitions Coverage

Similar to state coverage, the transitions play an important role in the overall coverage metric of power states. It is necessary to ensure that all valid transitions are covered and invalid ones (described via the UPF command describe_state_transition) are highlighted. In some cases, the unexpected behavior might happen due to some stuck-up supplies or issues in the power intent description. To catch such issues early in the implementation cycle, it is very important to provide stimulus exposing the concerned supplies through a series of power states to get the transitions covered.

C. Cross State Coverage

Low power designs have inter-connected power domains. These interconnections lead to various combinations of power states of these domains. A verification engineer should make sure that proper stimulus is applied to trigger these power state combinations. Because power states affect the placement of power management cells, it will also ensure verification of these power management cells. This is where cross state coverage (particularly between states added on such connected domains' supply) becomes relevant. Many static analyses are done on the basis of these cross states – for example, analysis to check the requirement of ISO/LS cell between two power domain boundaries. Coverage numbers for cross states would validate/invalidate the results of these static analyses. The relations between power states added on interacting power domains can be easily defined using the UPF command add_power_domain.

V. CHALLENGES IN MODELING COVERAGE METRICS

So far, we have been discussing the need of power state coverage metrics. We also defined three such metrics. Now, the question arises – How to describe and capture these power state coverage metrics? The Unified Coverage Interoperability Standard (UCIS) [2] defines various standard metrics related to coverage items. However, it does not provide any metric to capture power intent (power states, etc.). Moreover, power states themselves have the following unique features with respect to coverage modeling:

- Their coverage metrics require asynchronous sampling
- More than one power state can be true at a time
- The user can specify a state as illegal
- The power states of a UPF object can refer to the power states of other UPF objects.

These requirements can be easily met through SystemVerilog functional coverage features. The covergroup construct of SystemVerilog samples the activities of a signal/property at desired sampling points and one can distribute these sample points under different coverpoints and bins. These coverpoints and bins can be effectively used to collect coverage numbers of power states and their transitions. Covergroups also offer a generous syntax to handle a wide range of complex sampling scenarios especially the wildcard feature for handling state transitions when multiple states are true at the same time.

While modeling power state coverage metrics using covergroups, one faces a number of challenges:

1. How to access handles of power states or the objects where these states have been added? Power states can refer to following objects of design/UPF in their supply and logic expressions:
   - Design controls
   - Supply Ports and Nets created in the UPF/Design
   - Power Domains and their Power States
   - Supply Sets and their states
2. How to describe coverpoints and bins to capture coverage of power states and their transitions using the handles obtained in 1?
3. How to incorporate these power state metrics in the user's Design/Test?

We now propose a modeling style using covergroups and UPF's bind_checker command to address these challenges.

VI. Coverage Methodology

In the proposed methodology, all power state objects such as Power Domains and Supply sets have corresponding checker modules. These checker modules have covergroups and the rest of the logic required to model various power state coverage metrics. These checker modules would be inserted into the design using bind checkers. Now, we present a detailed description of various steps of the methodology. For illustration purposes, we will use the following add_power_state command as an example:

```
  add_power_state PD_LEAF2.primary \
      -state ON  {-supply_expr {power == {FULL_ON, 1.21} && ground ==
{FULL_ON,0.0} } } \
      -state OFF {-supply_expr {power == OFF    || ground == OFF} }
```

Here, PD_LEAF2 is a power domain defined in scope /top/leaf2. Two power states – ON and OFF – are added over its primary supply using add_power_state command.

A. *Binding Coverage Models to Design*

As discussed above, checker modules contain the actual coverage metric logic. This requires handles of various RTL and UPF objects referenced in the power state description (i.e. –logic_expr and –supply_expr of the add_power_state command). These handles are declared as ports of these checker modules. Also, these checker modules need to be inserted into the design hierarchy. The place of insertion will be the scope where a power state object has been defined in UPF.

Our modeling uses the UPF bind_checker command for accessing various objects and inserting the checker module at the appropriate places. The bind_checker command inserts checker modules into a design without modifying the design code or introducing functional changes. The IEEE 1801-UPF LRM defines bind_checker syntax in the following manner:

```
  bind_checker instance_name
  -module checker_name
  [-elements element_list]
  [-bind_to module [-arch name]]
  [-ports {{port_name net_name}*}]
```

Here, “instance_name” is an instance of the checker module - “checker_name”. “module” is the SystemVerilog module or VHDL entity/architecture for which all instances are the target of this command. “element_list” is the list of design elements where the instance will be inserted. “-ports” option maps the UPF/design signals to ports of the checker module. For our simple example, the checker module interface will look as following:

```
  module cov_PD_LEAF2_primary_PS (power, ground);
       input supply_net_type power;
       input supply_net_type ground;
```

The following bind_checker command will insert the checker module into the target design scope (/top/leaf2) and connect the checker module ports with UPF supplies PD_LEAF2.primary.power and ground PD_LEAF2.primary.ground:

```
  bind_checker PD_LEAF2_primary_PS_coverage
  -module cov_PD_LEAF2_primary_PS
  -elements {/top/leaf2}
  -ports {{power PD_LEAF2.primary.power} {ground PD_LEAF2.primary.ground}}
```

B. *Creating State determining logic to Capture Coverage*

The Coverage sampling and collection logic needs to have a representation of the active/inactive status of all power states. Now, using the ports of checker modules, we generate SystemVerilog Boolean expressions representing various power states. These expressions are assigned into different variables;we call these state variables. These state variables are used to collect state coverage information. In order to collect transition coverage information we define an array with state variables as elements; we call this an array transition variable. Any change in any of the state variables will trigger a clock which will act as the sampling event for state and transition coverage. State determining logic in our sample example looks like:

```
  wire [3:0] curr_state;

  reg cov_clk = 0;

   wire state_OFF = ((power.state == UPF::OFF) || (ground.state == UPF::OFF));

  wire  state_ON  =  (((power.state  ==  UPF::FULL_ON)  &&  (power.voltage   ==
1210000)) && ((ground.state == UPF::FULL_ON) && (ground.voltage ==  0.00)));
………..
   assign curr_state = {state_OFF, state_ON, ……};

   always @(state_OFF, state_ON, ……)
     cov_clk = 1'b1;

   always @(cov_clk)
     cov_clk = 1'b0;
```

C. *Defining Coverage Models using covergroups*

We define covergroups for state and transition coverage. The sampling event for these covergroups is the clock defined in section B. The covergroup modeling of the state coverage has a number of coverpoints that sample the state variables. The covergroup modeling of the transition coverage has a coverpoint with bins that sample various transitions of the transition variable. State and transition covergroups for our simple example appear as follows:

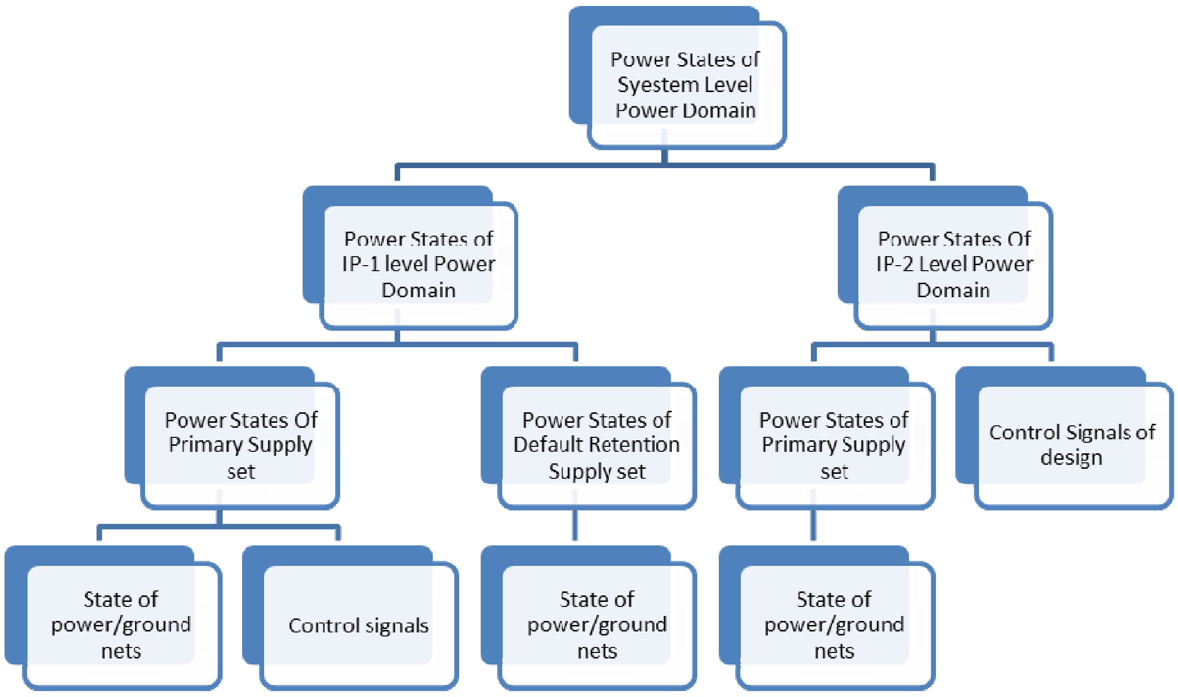
```
  covergroup primary_STATE_COVERAGE  @(posedge cov_clk);
           OFF: coverpoint state_OFF
           {
                bins ACTIVE  = (0=>1);
           }
           ON: coverpoint state_ON
           {
                bins ACTIVE  = (0=>1);
           }
       ……
        endgroup

        primary_STATE_COVERAGE PS_primary = new;

        covergroup primary_TRANSITION_COVERAGE  @(posedge cov_clk);
            type_option.strobe = 1;
            primary_TRANSITION_COVERAGE:coverpoint curr_state
            {
                wildcard bins OFF_to_ON  = (4'b???1=> 4'b??1?);
                wildcard bins OFF_to_DEFAULT_NORMAL  = (4'b???1=> 4'b?1??);
                wildcard bins OFF_to_DEFAULT_CORRUPT  = (4'b???1=> 4'b1???);
                wildcard bins ON_to_OFF  = (4'b??1?=> 4'b???1);
               ……
            }
        endgroup
```

VI. Example Design

In this section, we will explain our complete methodology using a small example.

A. Modeling Power States for a Sample Design

Power states are modeled typically in a hierarchical manner involving various supply sets and power domains. These states are defined via logic and supply expressions of the add_power_state command. A simple example describing the dependency tree of power states of various power domains and supply sets is provided in Figure 1. In this diagram, power states of a system level power domain depend upon power states of two IP level power domains (IP-1 and IP-2). Power states of power domain IP-1 depend upon power states of its primary and default_retention supply sets. Whereas power states of power domain IP-2 depend upon power states of its primary supply set and on some controls signals of the design. Now, coming to the next lower level of this dependency tree, power states of various supply sets depend upon state of supply nets and control signals. For example, power states of the primary supply set of power domain IP-1 depend upon states of power and ground nets along with some control signals.



Power States of Syestem Level Power Domain
Power States of IP-1 level Power Domain
Power States Of IP-2 Level Power Domain
Power States Of Primary Supply set
Power States of Default Retention Supply set
Power States of Primary Supply set
Control Signals of design
State of power/ground nets
Control signals
State of power/ground nets
State of power/ground nets

Figure 1

B. Capturing Power States in UPF

We will use an example to describe various details of the proposed modeling style. The hierarchical structure, power objects with power states, and various power states are pictorially represented in Figure 2. In this example, PD_TOP represents a system level power domain. It is defined in UPF file top.upf. PD_LEAF1 and PD_LEAF2 represent two IP level power domains. These are defined respectively in UPF files - ip1.upf and ip2.upf. Power states of PD_TOP (RUN, RET and SHD) depend on power states of PD_LEAF1 and PD_LEAF2. Power states of

PD_LEAF1 (RUN, RET, SHD) are dependent on power states of supply sets – PD_LEAF1.primary and PD_LEAF1.default_retention. Power states of PD_LEAF2 (RUN, SHD) are based on power states of PD_LEAF2.primary supply set. Power states of all the supply sets are derived from their power and ground supply nets. The relevant UPF commands corresponding to this structure are as following –

UPF file top.upf -

```
 create_power_domain PD_TOP -include_scope
 load_upf ip1.upf -scope leaf1
 load_upf ip2.upf -scope leaf2
 add_power_state PD_TOP \
    -state RUN {-logic_expr {leaf1/PD_LEAF1 == RUN && leaf2/PD_LEAF2 == RUN}}
\
    -state RET {-logic_expr {leaf1/PD_LEAF1 == RET && leaf2/PD_LEAF2 == RUN}}
\
    -state SHD {-logic_expr {leaf1/PD_LEAF1 == SHD && leaf2/PD_LEAF2 == SHD}}
```

UPF file ip1.upf -

```
 create_power_domain PD_LEAF1 -include_scope -supply {primary_sw_in}
 add_power_state PD_LEAF1.primary \
     -state ON  {-supply_expr {power == {FULL_ON, 1.21} && ground ==
{FULL_ON,0.0} } } \
      -state OFF {-supply_expr {power == OFF    || ground == OFF} }
 add_power_state PD_LEAF1.default_retention \
     -state ON  {-supply_expr {(power == {FULL_ON, 0.81} && ground ==
{FULL_ON, 0.0})}} \
     -state OFF {-supply_expr {power == OFF    || ground == OFF}}
 add_power_state PD_LEAF1 \
     -state RUN {-logic_expr {primary == ON && default_retention == ON }} \
     -state RET {-logic_expr {primary == OFF && default_retention == ON }} \
     -state SHD {-logic_expr {primary == OFF && default_retention == OFF}}
```

UPF file ip2. Upf -

```
 create_power_domain PD_LEAF2 -include_scope -supply {primary_sw_in}
 add_power_state PD_LEAF2.primary \
     -state ON  {-supply_expr {power == {FULL_ON, 1.21} && ground ==
{FULL_ON,0.0} } } \
     -state OFF {-supply_expr {power == OFF    || ground == OFF} }
 add_power_state PD_LEAF2 \
     -state RUN {-logic_expr {primary == ON}} \
      -state SHD {-logic_expr {primary == OFF}}
```

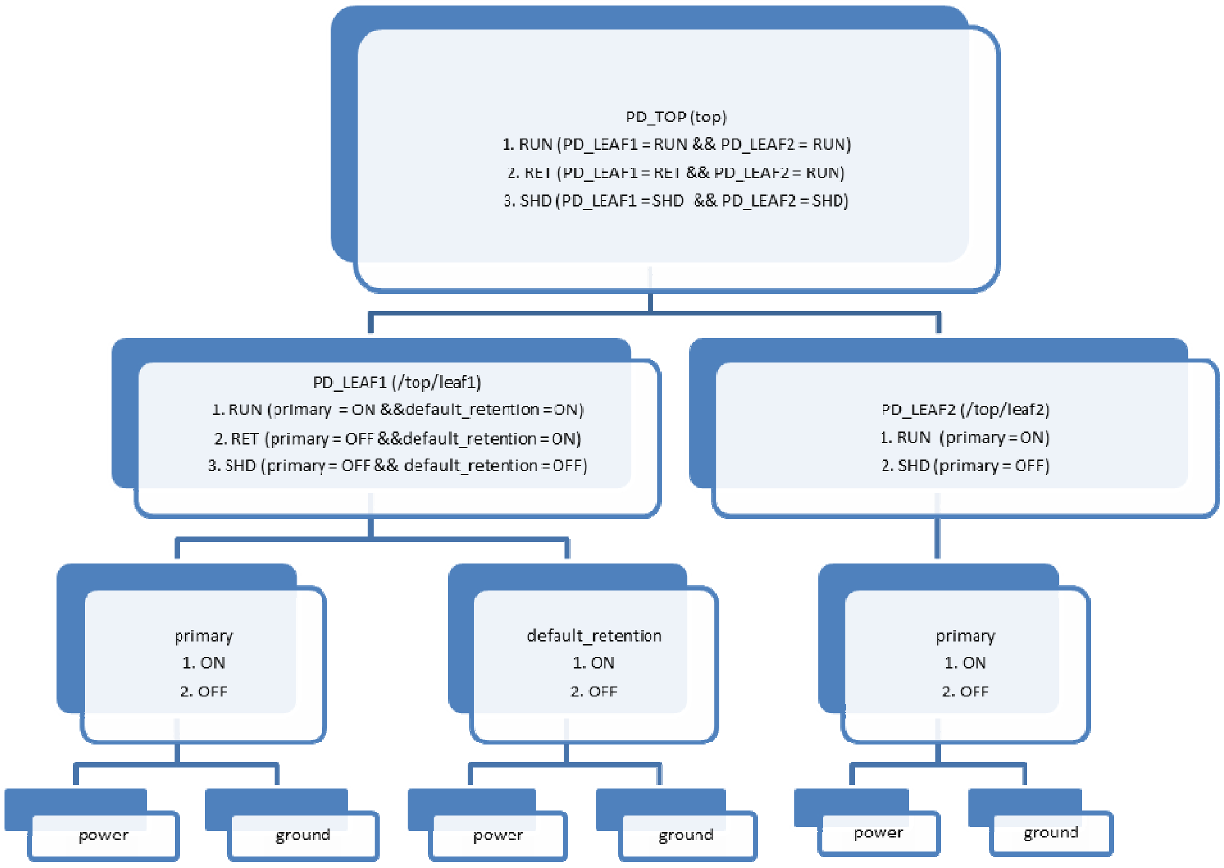

PD_TOP (top)
1. RUN (PD_LEAF1 = RUN && PD_LEAF2 = RUN)
2. RET (PD_LEAF1 = RET && PD_LEAF2 = RUN)
3. SHD (PD_LEAF1 = SHD && PD_LEAF2 = SHD)
PD_LEAF1 (/top/leaf1)
1. RUN (primary = ON &&default_retention = ON)
2. RET (primary = OFF &&default_retention = ON)
3. SHD (primary = OFF && default_retention = OFF)
PD_LEAF2 (/top/leaf2)
1. RUN (primary = ON)
2. SHD (primary = OFF)
primary
1. ON
2. OFF
default_retention
1. ON
2. OFF
primary
1. ON
2. OFF
power
ground
power
ground
power
ground

Figure 2

C. Coverage Models

In our example, the checker module corresponding to supply set PD_LEAF1.primary will be defined as follows:

```
module cov_PD_LEAF1_primary_PS (power, ground);
      input supply_net_type power;
      input supply_net_type ground;
      wire [3:0] curr_state;
      reg cov_clk = 0;
      wire state_OFF = ((power.state == UPF::OFF) || (ground.state ==
UPF::OFF));
      wire state_ON = (((power.state == UPF::FULL_ON) && (power.voltage ==
1210000)) && ((ground.state == UPF::FULL_ON) && (ground.voltage == 0.00)));
      wire state_DEFAULT_NORMAL = ((power.state == UPF::FULL_ON) &&
(ground.state == UPF::FULL_ON));
      wire state_DEFAULT_CORRUPT = !(state_ON || state_OFF ||
state_DEFAULT_NORMAL);

      assign curr_state = {state_OFF, state_ON, state_DEFAULT_NORMAL,
state_DEFAULT_CORRUPT};
      always @(state_OFF, state_ON, state_DEFAULT_NORMAL,
state_DEFAULT_CORRUPT)
```

```
            cov_clk = 1'b1;
        always @(cov_clk)
            cov_clk = 1'b0;

        covergroup primary_STATE_COVERAGE  @(posedge cov_clk);
            OFF: coverpoint state_OFF
            {
                 bins ACTIVE  = (0=>1);
            }
            ON: coverpoint state_ON
            {
                 bins ACTIVE  = (0=>1);
            }
            DEFAULT_NORMAL: coverpoint state_DEFAULT_NORMAL
            {
                 bins ACTIVE  = (0=>1);
            }
            DEFAULT_CORRUPT: coverpoint state_DEFAULT_CORRUPT
            {
                 bins ACTIVE  = (0=>1);
            }
        endgroup

        primary_STATE_COVERAGE PS_primary = new;

        covergroup primary_TRANSITION_COVERAGE  @(posedge cov_clk);
            type_option.strobe = 1;
            primary_TRANSITION_COVERAGE:coverpoint curr_state
            {
                wildcard bins OFF_to_ON  = (4'b???1=> 4'b??1?);
                wildcard bins OFF_to_DEFAULT_NORMAL  = (4'b???1=> 4'b?1??);
                wildcard bins OFF_to_DEFAULT_CORRUPT  = (4'b???1=> 4'b1???);
                wildcard bins ON_to_OFF  = (4'b??1?=> 4'b???1);
                wildcard bins ON_to_DEFAULT_NORMAL  = (4'b??1?=> 4'b?1??);
                wildcard bins ON_to_DEFAULT_CORRUPT  = (4'b??1?=> 4'b1???);
                wildcard bins DEFAULT_NORMAL_to_OFF  = (4'b?1??=> 4'b???1);
                wildcard bins DEFAULT_NORMAL_to_ON  = (4'b?1??=> 4'b??1?);
                wildcard bins DEFAULT_NORMAL_to_DEFAULT_CORRUPT  = (4'b?1??=>
4'b1???);
                wildcard bins DEFAULT_CORRUPT_to_OFF  = (4'b1???=> 4'b???1);
                wildcard bins DEFAULT_CORRUPT_to_ON  = (4'b1???=> 4'b??1?);
                wildcard bins DEFAULT_CORRUPT_to_DEFAULT_NORMAL  = (4'b1???=>
4'b?1??);
            }
        endgroup

        primary_TRANSITION_COVERAGE PS_TRANS_primary = new;
  endmodule
```

Here, power and ground correspond to the supply net power and ground whose states determine the power state of PD_LEAF1.primary.

Similarly, the skeleton of the checker module for PD_LEAF1 will look like:

```
  module cov_PD_LEAF1_PS ();
       wire [3:0] curr_state;
       reg cov_clk = 0;
```

```
	wire state_SHD = top.leaf1.PD_LEAF1_primary_PS_coverage.state_OFF && 
top.leaf1.PD_LEAF1_default_retention_PS"_coverage.state_OFF;
	wire state_RET = top.leaf1.PD_LEAF1_primary_PS_coverage.state_OFF && 
top.leaf1.PD_LEAF1_default_retention_PS"_coverage.state_ON;
	……
	assign curr_state = {state_SHD, state_RET, ……};
	always @(state_SHD, state_RET, ……)
	    cov_clk = 1'b1;
	always @(cov_clk)
	    cov_clk = 1'b0;

	covergroup PD_LEAF1_STATE_COVERAGE  @(posedge cov_clk);
	    SHD: coverpoint state_SHD
	    {
	         bins ACTIVE  = (0=>1);
	    }
	    ……
	    UNDEFINED: coverpoint state_UNDEFINED
	    {
	         bins ACTIVE  = (0=>1);
	    }
	endgroup

	PD_LEAF1_STATE_COVERAGE PS_PD_LEAF1 = new;

	covergroup PD_LEAF1_TRANSITION_COVERAGE  @(posedge cov_clk);
	    type_option.strobe = 1;
	    PD_LEAF1_TRANSITION_COVERAGE:coverpoint curr_state
	    {
	        wildcard bins SHD_to_RET  = (4'b???1=> 4'b??1?);
	        wildcard bins SHD_to_RUN  = (4'b???1=> 4'b?1??);
	        ……
	        wildcard bins UNDEFINED_to_RUN  = (4'b1???=> 4'b?1??);
	    }
	endgroup

	PD_LEAF1_TRANSITION_COVERAGE PS_TRANS_PD_LEAF1 = new;
endmodule
```

Here, power states of PD_LEAF1 are dependent on power states of supply sets PD_LEAF1.primary and PD_LEAF1.default_retention. So, we have used hierarchical references to state variables of PD_LEAF1.primary and PD_LEAF1.default_retention to define the state variables of PD_LEAF1.

D. Binding Coverage Models in User Design

The following bind_checker commands will be used to insert checker modules corresponding to PD_LEAF1. primary and PD_LEAF1 into their design scopes and make port connections:

```
bind_checker PD_LEAF1_primary_PS_coverage
-module cov_PD_LEAF1_primary_PS
-elements {/top/leaf1}
-ports {{power PD_LEAF1.primary.power} {ground  PD_LEAF1.primary.ground}}

bind_checker PD_LEAF1_PS_coverage
-module cov_PD_LEAF1_PS
-elements {/top/leaf1}
```

E. Simulation Results

The following is a sample textual report that prints various coverage metrics like state coverage and transition coverage.

```
# POWER STATE COVERAGE:
# -----------------------------------------------------------------------------------------------
-------
# UPF OBJECT                                            Metric      Goal/ Status
#                                                                At Least
# -----------------------------------------------------------------------------------------------
-------
#  TYPE : SUPPLY SET /top/leaf1/PD_LEAF1.primary         100.0%        100 Covered
#     Power State OFF                                   100.0%        100 Covered
#         bin ACTIVE                                         2          1 Covered
#     Power State ON                                    100.0%        100 Covered
#         bin ACTIVE                                         1          1 Covered
#     Power State DEFAULT_NORMAL                        100.0%        100 Covered
#         bin ACTIVE                                         2          1 Covered
#     Power State DEFAULT_CORRUPT                       100.0%        100 Covered
#         bin ACTIVE                                         1          1 Covered
#
#  TYPE : SUPPLY SET /top/leaf1/PD_LEAF1.primary          33.3%        100 Uncovered
#     Power State Transitions                            33.3%        100 Uncovered
#         bin OFF -> ON                                      1          1 Covered
#         bin OFF -> DEFAULT_NORMAL                          2          1 Covered
#         bin OFF -> DEFAULT_CORRUPT                         0          1 ZERO
#         bin ON -> OFF                                      1          1 Covered
#         bin ON -> DEFAULT_NORMAL                           0          1 ZERO
#         bin ON -> DEFAULT_CORRUPT                          0          1 ZERO
#         bin DEFAULT_NORMAL -> OFF                          1          1 Covered
#         bin DEFAULT_NORMAL -> ON                           0          1 ZERO
#         bin DEFAULT_NORMAL -> DEFAULT_CORRUPT              0          1 ZERO
#         bin DEFAULT_CORRUPT -> OFF                         0          1 ZERO
#         bin DEFAULT_CORRUPT -> ON                          0          1 ZERO
#         bin DEFAULT_CORRUPT -> DEFAULT_NORMAL              0          1 ZERO
#
#  TYPE : POWER DOMAIN /top/PD_TOP                        75.0%        100 Uncovered
#     Power State SHD                                   100.0%        100 Covered
#         bin ACTIVE                                         1          1 Covered
#     Power State RET                                     0.0%        100 ZERO
#         bin ACTIVE                                         0          1 ZERO
#     Power State RUN                                   100.0%        100 Covered
#         bin ACTIVE                                         1          1 Covered
#     Power State UNDEFINED                             100.0%        100 Covered
#         bin ACTIVE                                         2          1 Covered
#
#  TYPE : POWER DOMAIN /top/PD_TOP                        16.6%        100 Uncovered
#     Power State Transitions                            16.6%        100 Uncovered
#         bin SHD -> RET                                    0          1 ZERO
#         bin SHD -> RUN                                    1          1 Covered
#         bin SHD -> UNDEFINED                              0          1 ZERO
#         bin RET -> SHD                                    0          1 ZERO
#         bin RET -> RUN                                    0          1 ZERO
#         bin RET -> UNDEFINED                              0          1 ZERO
#         bin RUN -> SHD                                    0          1 ZERO
#         bin RUN -> RET                                    0          1 ZERO
#         bin RUN -> UNDEFINED                              1          1 Covered
#         bin UNDEFINED -> SHD                              0          1 ZERO
#         bin UNDEFINED -> RET                              0          1 ZERO
#         bin UNDEFINED -> RUN                              0          1 ZERO
#
# TOTAL POWER STATE COVERAGE: 62.5% POWER STATE COVERAGE TYPES: 12
```

VII. Conclusion

The coverage based methodology has become a critical part of the verification process. It is important to enable this methodology for power aware designs and the different objects created by UPF. In this paper, we have discussed the need of various state coverage metrics and highlighted the merits of each kind of coverage metric. The paper has also illustrated the importance of a customizable coverage modelling style for various state metrics. The prime focus of the paper has been to employ standard techniques to enable coverage based methodology for power states which uses the bind_checker feature of IEEE 1801-UPF. The techniques discussed in this paper can be used to create more complex coverage models that are customized as per design requirements. The technique can also be utilized by tools to automatically generate coverage models which can then be used for capturing the coverage of power states.

References

[1] IEEE Std 1801™-2013 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 29 May 2013.
[2] Unified Coverage Interoperability Standard Version 1.0 (UCIS), Accellera Systems Initiative Inc., June 2, 2012