Cross Coverage of Power States

Veeresh Vikram Singh
(veeresh_singh@mentor.com)
Awashesh Kumar
(Awashesh_kumar@mentor.com)
Mentor Graphics

Abstract-. **In today's low power SoC design flow, the role of power states has grown up enormously. Power states represent operating modes of a low power design and its various elements. These elements can often be inter-connected or related to each other. This relationship is also reflected in their power state descriptions. So, it becomes important that verification engineers verify occurrences of possible combinations/crosses of power states. Power states of a low power design can be defined using add_power_state command of UPF - the IEEE standard to specify power intent. These are often composed in a hierarchical manner, i.e. power states of a higher level power domain are dependent over power states of lower level domains. There is no readymade coverage metric available to capture the inter-dependence of hierarchical power states. Covering only the states and transitions of a particular system would not ensure that its possible combinations with the power states of subsystems have also been covered. To capture such information, a cross coverage metric is most suited. This paper presents a coverage-metric that captures occurrences of such inter-dependence to ensure that the system works as expected in all possible combinations of power states. It discusses in detail the challenges faced in modeling such metric and defines a generic and customizable methodology to capture cross-coverage of power states. A cross of power states can be huge in size if not defined with appropriate consideration. Cross-coverage in such scenarios would become difficult to interpret. This paper also touches on some cases where a cross coverage metric will not be appropriate.**

I. Introduction

Power states lie at the core of power aware verification. They capture various operational modes of a design. In order to ensure that a low power design is functionally correct, it becomes essential to ensure that all power states are covered in a comprehensive manner. Hence it becomes important to adopt a coverage-driven methodology for verification. However, success of verification depends on the comprehensiveness of the coverage metrics being used. Traditional code and functional coverage metrics would not suffice here because the power states are defined in the UPF (Unified Power Format) in an abstract manner. Hence, the need arises to bring out new low power specific coverage metrics.

In our previous paper [3], we had defined a set of coverage metrics to deal with low power verification. These metrics were related to power states and their transitions. However, with increasingly complex low power functionality, only covering states and transitions is not sufficient. A verification engineer also needs to know whether various systems and subsystems are in coherence or not. This necessity arises because various power management elements and power states are interdependent. So covering their simultaneous occurrence is also very important from verification point of view. In this paper, we would build on the methodology devised for state and transitions coverage of power states to define a new metric that captures these requirements. We name this metric as cross coverage of power states. In our opinion, this metric is more meaningful in capturing coverage of inter-dependent systems/subsystems. For the lack of space we would restrict the scope of this paper to power domains only. However, the same approach can be applied for supply sets also.

II. Power States In UPF

Power states represent supply information along with operational modes of design elements. IEEE Standard 1801-2013 Unified Power Format (UPF) [1] provides a set of commands that can be used to describe power states. These states can be associated with following UPF objects:

- Supply ports and nets
- Supply sets
- Power domains

A power domain is a collection of HDL instances that are treated as a group for power-management purposes. These instances are powered with the same supplies. Power states of such a domain can be defined in terms of the power states of supply sets associated with the domain. A power domain may sometimes represent an IP. This IP may contain a number of other power domains. Power states of a domain in this case can be defined in terms of power states of lower level power domains. Similarly, power domains may represent a system sometimes. This system may

be composed of a number of subsystems. All these subsystems will have a power domain associated with them. Here, power states of a system can be defined in terms of power states of various subsystems.

To represent power states in such a manner, UPF allows a power state definition of a power domain to reference the power state of any power domain or supply set, or the port state of any supply port or supply net, that is declared in the descendant subtree of the scope of that power domain. 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]
```

The add_power_state command provides the capability to add power states on supply sets and power domains. object_name is the name of supply set/power domain and state_name is the name of power state being added/updated. Various conditions and dependencies are described using–logic_expr and –supply_expr options. These options accept a Boolean expression. These Boolean expressions can have the power states of lower level subsystems. This helps in creating a hierarchical representation of power states similar to corresponding power domains.

III. Why Cross Coverage Of Power States?

A verification engineer is looking for answers to a number of questions while performing low power verification. Some important ones are:

1. All the desired power states and corresponding transitions reached or not?
2. If any illegal state or transition was reached?

These are some basic requirements and have been discussed in [3]. Now, at system level, simply knowing whether all valid states and transitions have been hit would not suffice. Here, power states are generally composed in a hierarchical manner i.e. power states of a higher level power domain are dependent over lower level power domains. A verification engineer will also be interested in knowing if all subsystems (lower level power domains) are interacting properly with the parent system/subsystem (higher level power domains). He is looking for a metric that captures all valid combinations of these inter-dependent power states. In other words, he is trying to ensure that all valid simultaneous occurrences of power states of these power domains have been covered. A cross coverage metric will provide a solution to all such requirements. Such a metric is also important because power states affect the placement of power management cells and coverage of their combinations will also ensure verification of these power management cells. Again, many static analyses are done on the basis of these cross states – for example, analysis to check the requirement of ISO/LS cell across power domain boundaries. Coverage numbers for cross states would validate the results of these static analyses.

IV. Challenges In Modeling Cross Coverage Metric

The Unified Coverage Interoperability Standard (UCIS) [2] defines various standard metrics for coverage items. Unfortunately, this standard does not provide any metric to capture coverage of low power objects such as power states. So, a verification engineer needs to devise his own metric using the existing UCIS metrics. Before selecting an existing coverage metric to model cross coverage of power states, we need to consider some peculiar properties of power states:

- 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.

Now, power states are defined in an abstract manner in UPF and the coverage model has to be written on one of the HDL languages. This again poses 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 define models to capture coverage of power states using the handles obtained in 1?
3. How to incorporate these power state metrics in the user's Design/Test?

In [3], we have discussed these challenges and proposed a solution using SystemVerilog covergroup constructs to overcome these. A brief description of the method proposed in that paper is given here for ready reference.

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.

A. Binding Coverage Models to Design

The checker module 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. These checker modules need to be inserted in the scope where a power state object has been defined. We are using UPF bind_checker command for accessing various objects and inserting the checker module at the appropriate places. 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.

B. Creating State determining logic to Capture Coverage

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 variables "state variables". These state variables are used to collect state coverage information. Any change in any of the state variables will trigger a clock which will act as the sampling event for state coverage.

C. Defining Coverage Model using covergroups

We define covergroups for state 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.

V. CROSS COVERAGE METHODOLOGY

The cross coverage methodology that we are proposing builds on the methodology proposed for power states coverage in [3]. A cross between power states of various power domains is written inside a checker module. This checker module contains logic for cross coverage. We use bind_checker to insert the checker module inside the design hierarchy. For illustration purposes, we will use the following add_power_state commands :

```
add_power_state PD_SYS -state PD_SYS_on {
     -logic_expr {PD_SUBSYS1 == SUBSYS1_on && PD_SUBSYS2 == SUBSYS2_on}}
add_power_state PD_SYS -state PD_SYS_ret {
     -logic_expr {PD_SUBSYS1 == SUBSYS1_ret && PD_SUBSYS2 == SUBSYS2_off}}
add_power_state PD_SYS -state PD_SYS_off {
     -logic_expr {PD_SUBSYS1 == SUBSYS1_off && PD_SUBSYS2 == SUBSYS2_off}}
```

The cross, we are interested in, is between power states of PD_SYS, PD_SUBSYS1 and PD_SUBSYS2.

A. Checker Module

1) Module interface

The checker module corresponding to the cross coverage metric will have an empty port list.

2) Capturing state variables and sampling clock

In [3], we have mentioned checker modules representing state coverage of power states. In that paper we have defined a special variable called "state variable" that represents a particular power state. In order to define cross coverage, we need to access these state variables inside the checker module corresponding to the cross coverage. This is done using SystemVerilog hierarchical references. We define an array variable with the size equal to all the contributing power states. Each element of this array represents a contributing power state to the cross. The elements are assigned hierarchical references to corresponding state variables. We will call this array "cross state array". A change in any element of the cross state array would trigger a clock which will be the sampling event for coverage collection. The cross state array, corresponding hierarchical reference assignments and the sampling clock in our example will look like the following:

```
module cov_PD_SYS_PS_CROSS();
reg cov_clk = 0;
wire [0:7] curr_state;

// Hierarchical references to state variables
assign curr_state[7] = top.PD_SYS_PS_COVERAGE.state_SYS_on;
assign curr_state[6] = top.PD_SYS_PS_COVERAGE.state_SYS_ret;
assign curr_state[5] = top.PD_SYS_PS_COVERAGE.state_SYS_off;
assign curr_state[4] = top.PD_SUBSYS1_PS_COVERAGE.state_SUBSYS1_on;
assign curr_state[3] = top.PD_SUBSYS1_PS_COVERAGE.state_SUBSYS1_ret;
assign curr_state[2] = top.PD_SUBSYS1_PS_COVERAGE.state_SUBSYS1_off;
assign curr_state[1] = top.PD_SUBSYS2_PS_COVERAGE.state_SUBSYS2_on;
assign curr_state[0] = top.PD_SUBSYS2_PS_COVERAGE.state_SUBSYS2_off;

always @(curr_state)
begin
    cov_clk = 1'b1;
end

always @(posedge cov_clk)
begin
    cov_clk = 1'b0;
end
```

Here, curr_state is the cross state array. Various hierarchical references are to the state variables corresponding to power states. The cross state array is actually representing the status of all power states contributing to the cross.

B. Cross Coverage Logic

We use SystemVerilog covergroup to model cross coverage of power states. The covergroup model of cross coverage has a coverpoint with bins that sample various combinations of contributing power states. These

combinations are tracked through the transition of cross state array from one combination to another. The sampling event here is the clock defined in section A(2). Cross coverage covergroup in our example looks like:

```
Covergroup PD_SYS_PD_SUBSYS1_PD_SUBSYS2_CROSS_COVERAGE @(posedge cov_clk);
         PD_SYS_PD_SUBSYS1_PD_SUBSYS2_CROSS_COVERAGE : coverpoint curr_state
         {
             wildcard           bins           \PD_SYS:PD_SYS_on-PD_SUBSYS1:SUBSYS1_on-
PD_SUBSYS2:SUBSYS2_on = (8'b???????? => 8'b1??1??1?);
             wildcard          bins          \PD_SYS:PD_SYS_ret-PD_SUBSSY1:SUBSYS1_ret-
PD_SUBSYS2:SUBSYS2_off = (8'b???????? => 8'b?1??1??1);
             wildcard          bins          \PD_SYS:PD_SYS_off-PD_SUBSYS1:SUBSYS1_off-
PD_SUBSYS2:SUBSYS2_off = (8'b???????? => 8'b??1??1?1);
         }
     endgroup
```

This covergroup represents the cross coverage model for power states of PD_SYS, PD_SUBSYS1 and PD_SUBSYS2. The wildcard bins are representing various combinations of power states defined using add_power_state. Here, a verification engineer can easily create only those bins that represent the power state combinations he is interested in. We have used wildcard bins here because they enable us to track the transition of cross state array to a combination without worrying about the previous combination or the power states that are not relevant for a particular combination.

C. Inserting coverage model into design

We rely on bind_checker command to insert the coverage model into the design. The syntax of bind_checker command is already defined in section IV(A).

In the following section, we will explain our methodology using a small example.

VI. EXAMPLE

A. Modeling Power States for a Sample Design

Power states are typically modeled in a hierarchical manner and refer to power states of various supply sets and power domains. These states are defined via logic and supply expressions of the add_power_state command. Figure 1 depicts a simple example describing the dependency tree of power states of a system level power domain. Power states of a system level power domain depends on the power states of the constituent subsystems Subsys1 and Subsys2. The power states of Subsys1 depend on the power states of its primary and retention supplies. Whereas power states of Subsys2 depend on power states of its primary supply set and some controls signals of the design. The power states of supply sets depend on the states of power and ground supply nets (part of –supply_expr) and control signals (part of the -logic_expr).

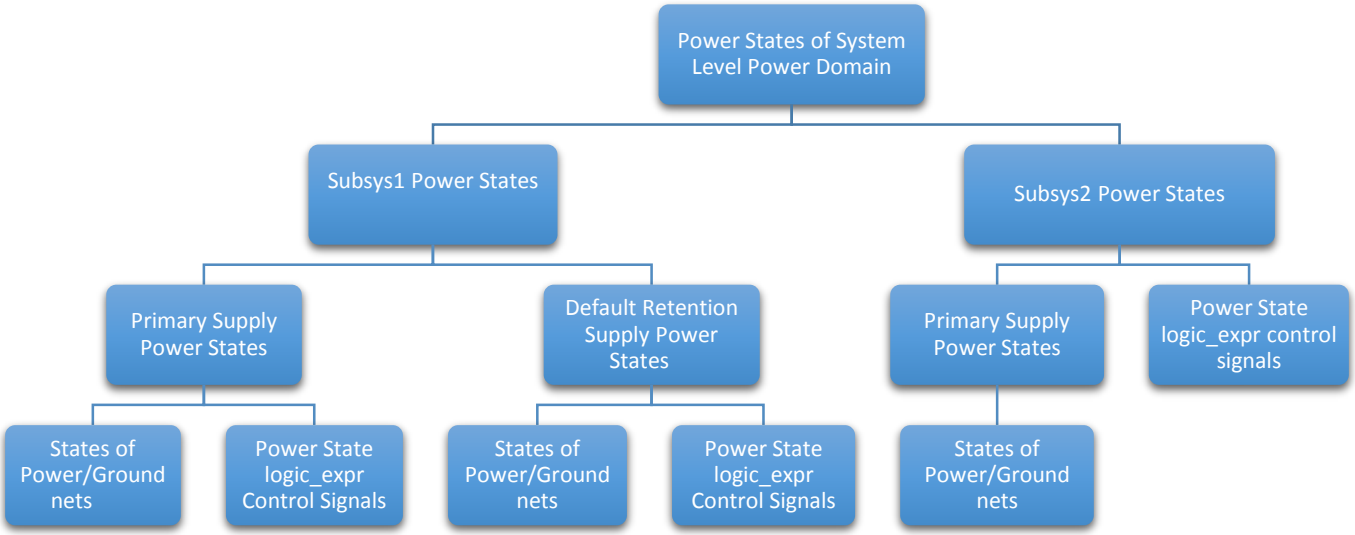

Figure 1. Example representing a hierachical description of Power States

B. Capturing Power States in UPF

We will use an example to describe the modeling style mentioned above. Figure 2 provides a pictorial representation of this modeling. Here, PD_SYS represents a system level power domain. PD_SUBSYS1 and PD_SUBSYS2 represent two lower level power domains. These might represent IP blocks in some cases. Power states of PD_SYS (PD_SYS_on, PD_SYS_ret and PD_SYS_off) depend on power states of PD_SUBSYS1 and PD_SUBSYS2. PD_SUBSYS1 can be in three power states: SUBSYS1_on, SUBSYS1_ret and SUBSYS1_off. These supplies are further dependent over power states of corresponding supply sets and so on. In our opinion, cross coverage is not a useful measure for supply sets. So, we will not go into details of supply set representation. Similarly, PD_SUBSYS2 has two power states: SUBSYS2_on and SUBSYS2_off. The relevant UPF commands corresponding to this structure are as following

```
add_power_state PD_SUBSYS1 -state SUBSYS1_on {....}
add_power_state PD_SUBSYS1 -state SUBSYS1_ret {....}
add_power_state PD_SUBSYS1 -state SUBSYS1_off {....}

add_power_state PD_SUBSYS2 -state SUBSYS2_on {....}
add_power_state PD_SUBSYS2 -state SUBSYS2_off {....}

add_power_state PD_SYS -state PD_SYS_on
     {-logic_expr {PD_SUBSYS1 == SUBSYS1_on && PD_SUBSYS2 == SUBSYS2_on}}
add_power_state PD_SYS -state PD_SYS_ret
     {-logic_expr {PD_SUBSYS1 == SUBSYS1_ret && PD_SUBSYS2 == SUBSYS2_off}}
add_power_state PD_SYS -state PD_SYS_off
     {-logic_expr {PD_SUBSYS1 == SUBSYS1_off && PD_SUBSYS2 == SUBSYS2_off}}
```

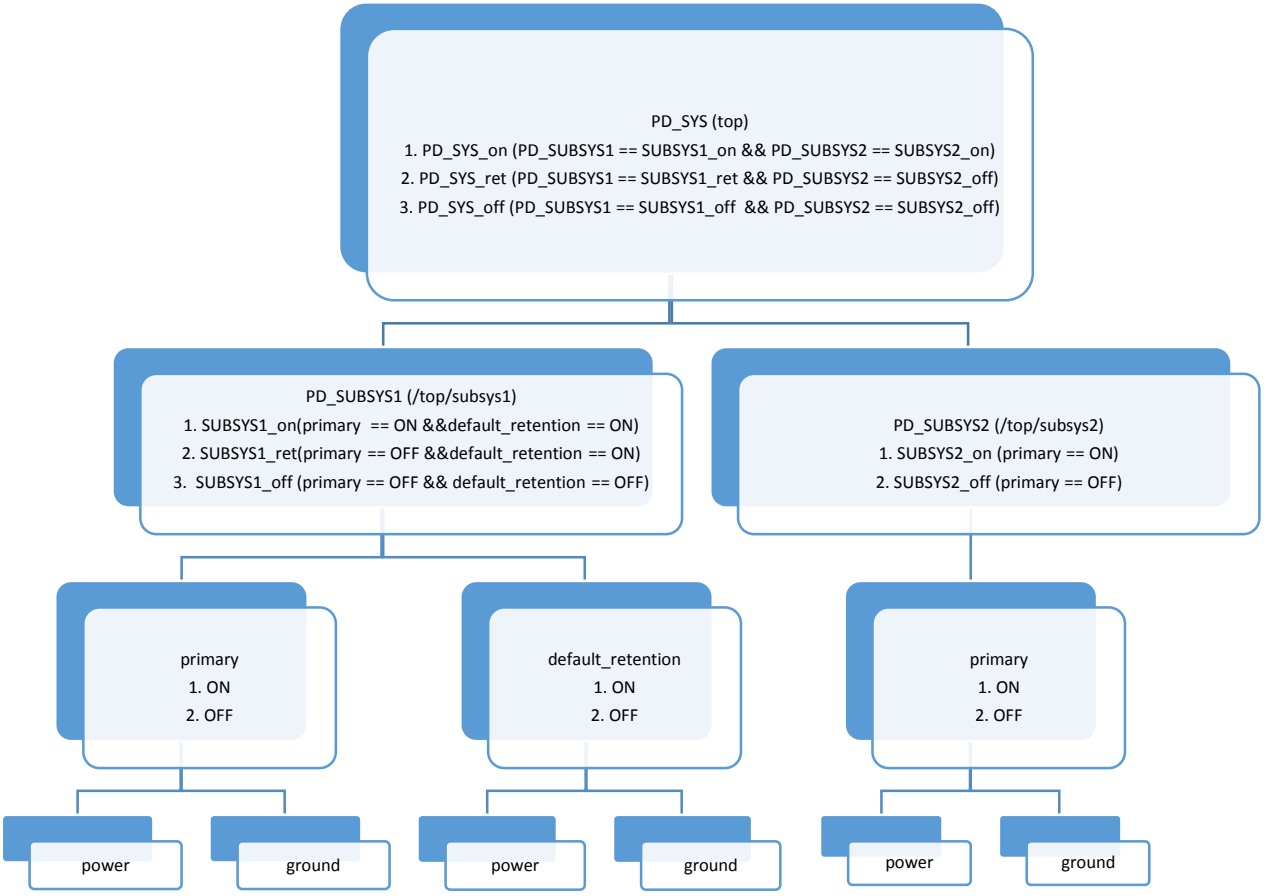

Figure 2. Power State Hierarchy

C. Coverage Models

In our example, the checker module corresponding to cross coverage of power states of PD_SYS, PD_SUBSYS1 and PD_SUBSYS2 would look like following:

```
module cov_PD_SYS_PS_CROSS();
  reg cov_clk = 0;
  wire [0:7] curr_state;

  assign curr_state[7] = top.PD_SYS_PS_COVERAGE.state_SYS_on;
  assign curr_state[6] = top.PD_SYS_PS_COVERAGE.state_SYS_ret;
  assign curr_state[5] = top.PD_SYS_PS_COVERAGE.state_SYS_off;
  assign curr_state[4] = top.PD_SUBSYS1_PS_COVERAGE.state_SUBSYS1_on;
  assign curr_state[3] = top.PD_SUBSYS1_PS_COVERAGE.state_SUBSYS1_ret;
  assign curr_state[2] = top.PD_SUBSYS1_PS_COVERAGE.state_SUBSYS1_off;
  assign curr_state[1] = top.PD_SUBSYS2_PS_COVERAGE.state_SUBSYS2_on;
  assign curr_state[0] = top.PD_SUBSYS2_PS_COVERAGE.state_SUBSYS2_off;

  always @(curr_state)
```

```
    begin
        cov_clk = 1'b1;
    end

    always @(posedge cov_clk)
    begin
        cov_clk = 1'b0;
    end

    covergroup PD_SYS_PD_SUBSYS1_PD_SUBSYS2_CROSS_COVERAGE @(posedge cov_clk);
      PD_SYS_PD_SUBSYS1_PD_SUBSYS2_CROSS_COVERAGE : coverpoint curr_state
        {
               wildcard          bins          \PD_SYS:PD_SYS_on-PD_SUBSYS1:SUBSYS1_on-
PD_SUBSYS2:SUBSYS2_on = (8'b???????? => 8'b1??1??1?);
               wildcard         bins         \PD_SYS:PD_SYS_ret-PD_SUBSSY1:SUBSYS1_ret-
PD_SUBSYS2:SUBSYS2_off = (8'b???????? => 8'b?1??1??1);
               wildcard         bins         \PD_SYS:PD_SYS_off-PD_SUBSYS1:SUBSYS1_off-
PD_SUBSYS2:SUBSYS2_off = (8'b???????? => 8'b??1??1?1);

         }
     endgroup
     PD_SYS_PD_SUBSYS1_PD_SUBSYS2_CROSS_COVERAGE  PS_CROSS_PD_SYS_PD_SUBSYS1_PD_SUBSYS2
= new;
  endmodule
```

D. Binding Coverage Models in User Design

The following bind_checker commands will be used to insert checker modules corresponding to cross coverage model of PD_SYS, PD_SUBSYS1 and PD_SUBSYS2 into the design:

```
  bind_checker PD_SYS_PS_CROSS_COVERAGE \
       -module cov_PD_SYS_PS_CROSS \
       -elements {/top}
```

VII. LIMITATIONS OF CROSS COVERAGE MODEL

The example that we considered here is simple one with power states defined in a hierarchical manner using only == and && operators. Such power states are definite power states [4]. However, if these states are defined using some other operators such as ||, != or !, the possible combinations of inter-dependent power states will become huge in number. Such power states are called indefinite power state. In such cases, it would be difficult to create and track cross coverage metric. Hence, the model is recommended to be used only for definite power states.

VIII. CONCLUSION

Coverage-driven verification is frequently used in non-power aware designs. In order to reap its benefits for power aware designs, it is important to devise various coverage metrics for such designs. In this paper, we designed a cross coverage metric that provides a generic, customizable approach to capture the coverage of inter-dependent power

states. We discussed the challenges with the existing approaches to capture such inter-dependence. We also demonstrated with relevant examples that why only state and transition coverage metrics will not be sufficient to capture various operating modes of a system. We have provided the customizable RTL cross coverage model using UPF and SystemVerilog constructs. This model can be extended to more complex power aware designs.

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
[3] "Amit Srivastava, Pankaj Kumar Dwivedi, Veeresh Vikram Singh", Let's DisCOVER Power States, DVCon USA 2015
[4] "Erich Marschner, John Biggs", Unleashing the Full Power of UPF Power States, DVCon USA 2015