# Low Power Coverage: The Missing Piece in Dynamic Simulation

Progyna Khondkar, Gabriel Chidolue and Ping Yeung
Mentor Graphics, a Siemens Business, Fremont, CA, USA and Newbury, UK.

***Abstract*- Low power (LP) design verification is an emerging technology, and almost every chip design today incorporates UPF (IEEE-1801 standard) based power dissipation and reduction techniques to manage and control the power on chip. Coverage data from LP verification in general originates from UPF and relevant HDL objects, i.e. power domains, power supplies, power states, different power strategies, control signals and ports of the power strategies. Obviously the nature of power states and their transitions are quite different from non-LP state machines. In addition, the industry lacks in semantic references for formation of power state machines or power states and state-transition to develop a complete LP coverage computation models, as well an adaptable database with application programming interface (API) to collect, access and represent the computed LP coverage. In order to fulfill these missing pieces, we first identified all the resources of the LP coverage contributors and categorized them as *UPF cover-bins*. We also identified *UPF cross-cover-bins* for interdependent power states in a complex hierarchical UPF flow and proposed a simplified dependency graph to represent interdependent power states coverage computation models. Through real design examples and case studies, we demonstrate how to achieve comprehensive LP design verification closure with all possible sources of power states, their transition coverage and cross-coverage of power domains of interdependent states. As well the paper also proposes the mechanism to combine and represent LP and non-LP coverage in a unified and adaptable database with API accessibility.**

## I. INTRODUCTION

Dynamic simulation results are inconclusive in nature and often need to be quantified with definitive metrics that possibly denote verification coverage closure through numeric values (in percentages) in conjunction with appropriate design parameters. Coverage provides meaningful insight into design verification completeness. The coverage metric in dynamic simulation is a system or standard of measurement used to describe the degree to which the design is exercised with certain design objects or parameters for a particular test suite or testplan execution. Even the testplan is subject to measurement as a weighted metric and recapitulated to contribute to the total resultant coverage metric for the design. The resultant metrics from such diversified objects or parameters are stored in a common, unified coverage database (UCDB). UCDB provides accessibility to further enhance the coverage metrics with new coverage results from different new sources through coverage merging, as well as a mechanism to analyze and generate the coverage reports through API, such as the industry standard Accellera UCIS API.

Unlike non-LP coverage, LP coverage data solely originates from the abstraction of UPF and relevant HDL objects. Moreover, in low power dynamic simulation state space, the UPF power states and their transitions are asynchronous in nature and may refer or depend on other power states. Even more than one power state can remain true at a time, while it is possible to mark any power state as illegal anytime. These unique but contradictory features of power states with respect to non-LP state machines make it difficult to formulate LP coverage computation models and coordinate with UCDB. Obviously the coverage information extraction from power states and their transitions are difficult and must rely on an exhaustive process. The characteristics of a power state are diversified in nature and can be best summarized as follows.

**List 1:** Characteristics of Power States

- Power states are abstract at higher levels and physical (supply port and nets) at lower levels of design abstraction,
- They are applicable for different UPF objects that include rudimentary parts of supply networks and design elements; e.g. power supplies, power domains, design groups, design models, and design instances,
- Power states may reference descendant power domains or power supply states from the scope of top domains,

- Power states denote different operation modes based on different combinations of power domains and their power supplies,
- They are subject to interdependency between different UPF objects; e.g. power domain and supply set
- It is possible that power states may be exposed to spontaneous state transitions between interdependent objects.

Hence, it is required to comprehend all possible sources of power states and their transitions from the above lists for developing a complete LP coverage computation model.

### *A. Motivation: The Missing Pieces of LP Coverage*

UPF based Low power verification is now industry standard and mandatory part in the design verification and implementation flow (DVIF). Hence LP coverage became attractive topics to the low power verification community in recent years. Specifically, several papers have been published in the last few years (2015-17), focusing on power state and transition coverage features and other LP coverage functionalities based on UPF and HDL objects [3], [4], and [6].

However the foundation of low power coverage is not focused in a consolidated manner and remains detached from non-LP coverage. Because primary sources of LP coverage – the power states are abstractly defined in UPF. Consequently there is no pre-defined coverage methodology to capture power states and their transition. The UCIS also do not have any standard for low power coverage based on UPF intents at the time when this paper is written. Although EDA tools often craft SystemVerilog covergroup to sample different low power signals to collect different coverpoints and cover bins [6] with coverage properties — they are just inadequate to cover numerous sources of low power states and their transitions, as well accommodation or merging of LP coverage in UCDB with non-LP coverage. As a result, the LP coverage remains a missing piece in the entire functional verification environment, specifically in dynamic simulation.

In this paper, we are motivated with the objective to create the foundation of a comprehensive and standard LP coverage computation model to fulfill the missing pieces. We demonstrate LP coverage computation and representation methodologies with case studies and real examples that are actually incorporated and executed through LP dynamic verification tools. We further hope that the proposed methodology, standardization and experimental results from our studies will remain as the first source of physical interpretation and implementation proof of the complete LP coverage computation models and database.

### *B. Organization of This Paper*

This paper is organized in the following structure. In Section II, we conduct exhaustive studies to identify every possible contributor that forms the foundation of low power coverage. This section also explains the LP coverage methodologies with specific examples that are defined and as well that are missing in the UPF language reference manual (LRM) [1], [2]. The next section explains the comprehensive LP coverage computation models through case studies. The proposed LP coverage computation models — an adaptive coverage database to fulfill the missing pieces are explained in Section III. The final Sections IV shows the prospect of further research and shares concluding remarks. References are shown at the end.

## II. FOUNDATION OF LP COVERAGE

In order to propose a complete LP coverage solution, at first we will explain the fundamental concept of UPF intents on a simple design. Then we find out the primary contributors of LP coverage from the characteristics of power states as extracted from UPF methodologies and as explained in Section I.

Let us consider a simple CPU core design to apply UPF intent on it. **Figure 1** explains multiple power domains (PD) on the CPU core (like PD_top, PD_sub1 through PD_sub5 etc.) which confines different portions of the design. For simplicity, the examples and explanations used throughout this paper, we will only consider the default PD_top, PD_sub1 and PD_sub2 power domains, which are for the cpu_top module and uALU_top and uALU_sub instances of the design respectively. Here, the PD_top has only one- ON power state, while PD_sub1 and PD_sub2 both have ON and OFF power states. In addition, PD_sub1 also possesses state retention registers (RFF) to store the states and data during power OFF. In addition, the **Figure 1** also shows most of the special LP or multi-voltage (MV) cells, like isolation (ISO) cells, enable-level-shifter (ELS) cells (which are combined with ISO and LS ), power-switch (PSW) cells and level-shifter (LS) cells.

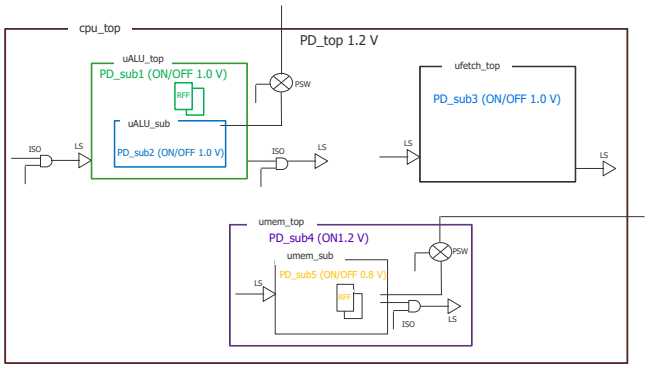


**Figure 1 Concepts of UPF, Power Domain and UPF Strategies (ISO, ELS, PSW, RFF) on a Design**

Regardless of UPF versions and releases (UPF 1.0/2/2.1/3/3.1), we identified that the power-state machines (power states and their transitions) in an LP simulation (LP-SIM) or coverage analysis environment can originate from one or a combination of the UPF constructs, UPF commands and their relevant options shown in **List 2**.

*C. Contributors of LP Coverage*

**List 2:** The Source of Power States and Their Transitions from UPF Constructs
- Supply Port States from **add_port_state,**
- Supply Net States from Power State Table (PST),
- PST States from **add_pst_state**,
- Power Domain States from **add_power_state**,
- Supply Port, Supply Net, and Supply Set Function States from **add_supply_state,**
- Power States of the Power Supply Sets from **add_power_state,** etc.

It is evident that the power states originating from these different types of UPF constructs directly affect the requirements and placement of special power management cells, often known as multi-voltage (MV) or LP cells, such as ISO, ELS, PSW, and RFF in the design. Hence it is also required to collect coverage information from these MV cells. However, for dynamic simulation, the coverage information for MV cells, apart from PSW, may be collected only from the different states and transitions of their controls and acknowledgement signals. Specifically the signals can be listed as shown below.

**List 3:** The Source of Power States and Their Transitions from UPF Strategies
- Isolation “Enable” Signal,
- Retention “Save and Restore” Signals,
- Power Switch States and Transitions,
- Power Switch “Control Port”,
- Power Switch “Ack Port”

Recalling the syntax and example of ISO, RFF, and PSW from [2], it is evident that all of the above mentioned control and acknowledgement signals have the following transitions:

**List 4:** Transitions of Control Signals for UPF Strategies
- High-to-Low and
- Low-to-High Transitions.

As well, during simulation, the status of these signals may remain in one of the following states:

**List 5:** States of Control Signals for UPF Strategies
- Active through presenting a value (level sensitive) or transition (edge sensitive),
- Inactive (opposite to the active)

- Active x (driving unknown)
- Active z (remain floating or un-driven)

In addition, apart from the control and acknowledgement signals, the functionality of the PSW itself allows the following state values and their possible combination of transitions as defined in the IEEE 1801 specification:

**List 6:** State Values of Power Switch, Control, and Acknowledge Ports
- ON state,
- OFF state,
- Partial ON state and
- UNDETERMINED (ERROR) state.

Here, at any given time, the ON or the partial ON state contributes a value to the output port of the PSW. The UNDETERMINED status comes into existence only when the ON or partial-ON state Boolean expression for a given input supply port, which is not in the OFF state, refers to an object with an unknown (X or Z) value, then the contributed value at the output of PSW remains {UNDETERMINED, unspecified}.

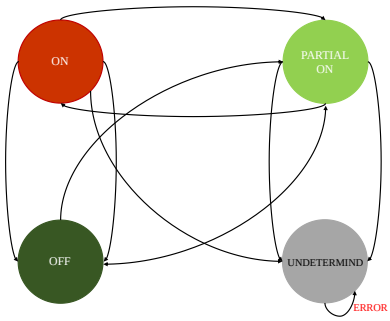


**Figure 2 Power state-Transition Diagram of PSW**

However, in LP-SIM, the UNDETERMINED states are interpreted as ERROR states. Hence the PSW itself also has the states as shown in **List 6**. The coverage metric required to cover all possible transitions between these states for a PSW is best represented in **Figure 2**.

Hence, from the above extractions and observations of UPF methodologies, objects, strategies and state values, we realized that the foundation of an LP coverage information model can be developed from power states and their transitions based on fundamental aspects of different UPF constructs from **add_port_state, add_pst_state**, **add_power_state** and **add_supply_state**. We define these set of UPF commands as UPF coverage constructs. Whenever a design encounters these UPF coverage constructs, the LP-SIM or coverage analytical engine will generate *UPF cover-bins*. We define *UPF cover-bins*, as shown below, as a combination of coveritem and coverpoint, based on UCIS models [7].

***UPF cover-bins****: This is a counter construct with specific decorated items. These items are generalized and based on UPF coverage constructs, i.e. name of state, status (legal/illegal), scope (design scope), attribute (ports or nets) etc. The UPF cover-bins represents LP coverage data collected from corresponding UPF coverage constructs.*

Obviously the items in **List 3** through **List 6**, which are the sources of additional power states and their transitions from power strategies, as well as transitions, states and state values of control (and acknowledge) signals of these strategies will also generate additional UPF cover-bins accordingly.

From the above discussion, it is apparent that designing a complete LP coverage computation model is complicated and has to rely intensely on various nonstandard and nontraditional aspects of coverage measurements. Specifically, explicitly decorating the items within an UPF cover-bin, for example, legality status, scope of coverage contribution in terms of design hierarchy, as well as some implicit features listed in **List 1**, such

as different operation modes of power states based on different combinations of power domains and their power-supplies, interdependency between different UPF objects; e.g. power domain and supply set and the possibility of power states to spontaneous state transitions between interdependent objects.

However, through a comprehensive and coherent analytical model, tool, and methodology augmentation; it is possible to completely enumerate the coverage collection, analysis, and results representation through UPF cover-bins. The entire LP coverage metric computation model mostly relies on the characteristics of power states shown in **List 1** through **List 6**. We further updated the list of LP coverage contributors to a new list based on the analytical discussions in the previous two sections and also considering few non-LP coverage features, like code coverage in LP environment.

**List 7:** Updated List for the Complete Sources of LP Coverage Computation Model
(1). Coverage information from LP Dynamic Checks based on;
- LP testbench and LP augmented RTL (Code Coverage),
- Automated LP Sequence Checkers and
- Custom LP Checkers,

(2). Coverage Information from Power States and Power State Transitions based on;
- Design controls,
- Supply ports and nets created in the UPF and design,
- Power domains and their power states,
- Supply sets and their states,
- Power Switch States and their Transitions,
- State transitions for ISO, RFF, PSW Control and Acknowledgement signals,

(3). Coverage Information from Cross-Power Domain Power States Dependency based on;
- All possible combinations of interdependent power states,
- As well as their possible spontaneous transitions.

We will discuss each of the subheader categories mentioned in the **List 7**, in succedding subsections.

*D. Coverage Information from LP Dynamic Checks*

The common assumption about custom LP checkers and automated LP sequence checkers are that they generally contribute to coverage when they have never failed and have passed at least once during dynamic simulation. Their coverage computation models are primarily based on either a SystemVerilog covergroup or its extension. However the LP or UPF augmented code coverage is not straightforward. UPF adds power artifacts and instruments initial begin blocks, always@ blocks sensitivity lists, and it changes the structures of design instances, branches and conditions. Hence, generating UPF cover-bins for UPF instrumented RTL code to compute LP code coverage and merging it later with non-LP code coverage is impractical. However, toggle coverage may still be applied on the UPF instrumented RTL code since it looks only for HDL objects with values such as variables and nets and tracks the changes in those values.

Hence for a complete RTL code coverage model, we propose that RTL code coverage must be accomplished in a non-LP environment and merge only LP toggle coverage data from UPF instrumented RTL code to non-LP code coverage results. Neverthless we also need to make sure that the other code coverage data models — specifically, branch, statement, blocks, condition, expressions etc. — for both UPF instrumented RTL and non-LP RTL remain formally or logically equivalent. However, formulating the code coverage data models equivalency checks are out of the scope of this paper and will be discussed in a different platform.

*E. Coverage Information from Cross-Power Domain Power States Dependency*

Since we have already discussed on coverage information from power states and power state-transitions in Section II-C, hence we will focus our discussion on coverage information from power domain power state dependency. In order to understand the dependency we need to clarify the UPF LRM semantics responsible for such dependency. UPF defines a power state for a power domain and its associated supply networks through **add_power_state** command. The definition further allows referencing the port state of any supply port or supply net in the descendant subtree from the scope of the top power domain. The UPF **add_power_state** syntax is shown below:

**Example 1** UPF Syntax for **add_power_state**
**add_power_state**
**[*-supply* | *-domain* | *-group* | *-model* | *-instance*]** object_name

**[-*update*]**
**[-*state* {**state_name **[-*logic_expr* {**boolean_expression**}] [-*supply_expr* {**boolean_expression**}]**
**[-*simstate* simstate]**
**[-*legal* | -*illegal*]}] [-*complete*]**

The **add_power_state** command provides the capability to augment power states to different UPF objects through the <object_name>. The <state_name> represents the user specified name for a power state that is just defined or updated for the <object_name>. The ***–logic_expr*** is used to define the power states for either supply sets or power domains. The expressions are constructed by referencing control conditions, clock frequencies, and power states of the domain supply sets. On the other hand ***-supply_expr*** is used only for power states of supply sets; it refers to the functions of supply states of the supply set. Each of these expressions can be further defined as either legal or illegal.

Hence it is distinctive that the logic and supply expressions in the **add_power_state** definition are based on various conditions through Boolean expressions. These Boolean expressions may contain control conditions, design parameters as well as power state information from different power domains including hierarchically lower level domains. Though such subtree power state referencing from top, allows symmetrical hierarchical representations of power domains, supply network, and their corresponding power states, but they may also impose inter-state dependency that is often difficult to track. The following simple example shows the power states of the PD_top domain in terms of logic expression.

**Example 2** UPF add_power_state Sample Example with *-logic_expr*
add_power_state PD_top -state SYS_ON {*-logic_expr* {PD_sub1 == SUBSYS1_ON && PD_sub2 == SUBSYS2_ON}}
add_power_state PD_top -state SYS_OFF {*-logic_expr* {PD_sub1 == SUBSYS1_OFF && PD_sub1== SUBSYS1_RET && PD_sub2 == SUBSYS2_OFF}}

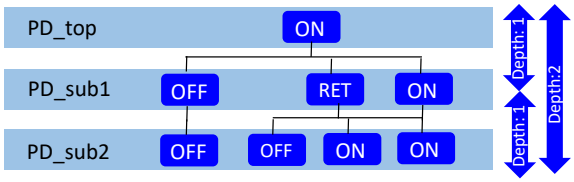


**Figure 3 Power state Dependency Diagram**

These examples are proof of dependency of power states among power domains from PD_top to subtree domains, PD_sub1 and PD_sub2, and it is further clarified in **Figure 3**.

Since these power states and transitions are highly interdependent in a hierarchical UPF flow, we define a new *UPF cross-cover-bins* for such dependent power states coverage. All though *cross-cover-bins* are just extensions of previously defined UPF cover-bins, they possess additional decoration items to determine the depth of hierarchical crossings.

The UPF 1801-2013 LRM (UPF 2.1) and 1801-2015 LRM (UPF 3.0) defines two semantically identical commands for monitoring the legality of power state transitions from one named power state to another; namely **describe_state_transition** and **add_state_transition**; but their semantics do not comply with the context and requirements to decorate the *UPF cross-cover-bins*.

Hence we propose an extension of the UPF methodology for computing cross-coverage power states and their transitions. We also developed an internal mechanism to decorate the cross-cover-bin to compute coverage for hierarchically dependent power domains. For simplicity, let us first explain the cross-cover-bin computing flow and then the extension of the methodology. The computation model of cross-cover-bins can be best represented by a simple dependency graph (**Figure 4**). The nodes of the graph are based on power state logic and supply expressions, where the nodes represent power states, and their edges represent transitions between the nodes. A path between the nodes denote a sequence of nodes and edges, connecting a node with its descendant and (or) dependent.

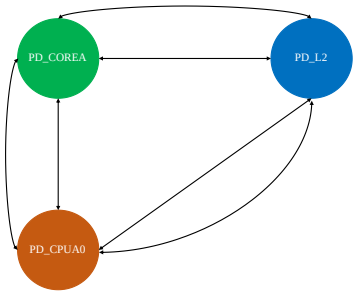


**Figure 4 Dependency Graph for Cross-Coverage Computation Model for Power Domains Group PD_top->PD_sub1->PD_sub2.**

The LP-SIM or coverage analytical engine are required to monitor and capture the transitions from the edges of the graph. The path provides the depth for a group of interdependent nodes and helps to untie the dependency among these nodes. Now the proposed UPF semantic, as shown and explained below, facilitates the coverage analytical tool to decorate the *cross-cover-bins.*

**Example 3** Methodology Extension for UPF Cross-Cover-Bin
**describe_state_cross_coverage**
[***-domains*** domains_list]
[***-depth*** cross_coverage_depth]

The **describe_state_cross_coverage** command is an augmentation of current UPF intent to supplement the missing cross-coverage power state-transition monitoring and coverage computation semantic model. The ***-domain*** defines the list of power domains for which cross-coverage needs to be computed. The ***-depth*** is the number of power domains that are involved in the cross-hierarchy and are interdependent. By default, the computation starts with depth one; however the coverage analytical tool will figure out the list of dependent power domains of a particular top or adjacent domains. These dependent power domains together with the specified top domain will form a domain group. Then the tool computes the *cross-cover-bins* or cross-coverage results for this particular group of power domains. Hence for the ***-logic_expr*** example of **add_power_state**, shown above, and from the theory shown in **Figure 4**, the *cross-cover-bins* will decorate accordingly for PD_top->PD_sub1->PD_sub2 power domain group as follows.

**TABLE 1** Dependent States for Power Domains

| Power Domains | **PD_top** | **PD_sub1** | **PD_sub2** |
|---|---|---|---|
| Power States | SYS_ON | SUBSYS1_ON | SUBSYS2_ON |
| Power States | SYS_ON | SUBSYS1_RET | SUBSYS2_ON |
| Power States | SYS_OFF | SUBSYS1_OFF | SUBSYS2_OFF |

**TABLE 2** Cross-Coverage Data for *-depth*=1 (default) for PD_top->PD_sub1->PD_sub2

| |
|---|
| SYS_ON -> SUBSYS1_ON -> SUBSYS2_ON |
| SYS_ON -> SUBSYS1_RET -> SUBSYS2_ON |
| SYS_OFF -> SUBSYS1_OFF -> SUBSYS2_OFF |

The coverage analytical tool specific details on coverage collection, analysis, and representation of the results for the UPF cross-cover-bin and UPF cover-bin are demonstrated in succeeding sections respectively.

*F. Case Study: Coverage Computation for UPF Cross-Cover-Bins*

Recalling **Figure 4** and **Example 3** in **Section II**, the dependency graph and extended methodology for collecting cross-coverage data through the **describe_state_cross_coverage** is explained with a specific UPF example containing logic expression in **add_power_state**.

**Example 4** Sample example for cross-coverage collection from **add_power_state**
**add_power_state** PD_OUT ***-state*** PD_OUT_on {***-logic_expr*** {PD_OUT***.primary*** == PD_OUT_primary_on}}

```
add_power_state PD_OUT -state PD_OUT_off {-logic_expr {PD_OUT.primary == PD_OUT_primary_off}}
add_power_state PD_OUT -state PD_OUT_ret {-logic_expr {PD_OUT.primary == PD_OUT_primary_off &&
PD_OUT.default_retention == PD_OUT_ret_on}}
add_power_state PD_OUT2 -state PD_OUT_on {-logic_expr {PD_OUT == PD_OUT_on}}
add_power_state PD_SUBSYS2 -state PD_SUBSYS2_on \
        {-logic_expr {PD_SUBSYS2.primary == PD_SUBSYS2_primary_on}}
add_power_state PD_SUBSYS2 -state PD_SUBSYS2_off \
        {-logic_expr {PD_SUBSYS2.primary == PD_SUBSYS2_primary_off}}
add_power_state PD_SUBSYS1 -state PD_SUBSYS1_on \
        {-logic_expr {PD_SUBSYS1.primary == PD_SUBSYS1_high_volt && PD_OUT2 == PD_OUT_on}}
add_power_state PD_SYS -state RUN \
        {-logic_expr {PD_SUBSYS1 == PD_SUBSYS1_on && PD_SUBSYS2 == PD_SUBSYS2_on}}
add_power_state PD_SYS -state SLEEP \
        {-logic_expr {PD_SUBSYS1 != PD_SUBSYS1_on && PD_SUBSYS2 != PD_SUBSYS2_on}}
### configure cross coverage ##
describe_state_cross_coverage -domains {PD_SYS} -depth 3
describe_state_cross_coverage -domains {PD_SUBSYS1} -depth 2
describe_state_cross_coverage -domains {PD_OUT2}
```

In the above examples, the **describe_state_cross_coverage** is given for three different power domains to find the cross-coverage data for different depths ranging from none, which is default, and depth 1 for PD_OUT2, depth 2 for PD_SUBSYS1, and depth 3 for PD_SYS. The tool will process the dependency graph for each category based on the depth information and extract the state transition from the **add_power_state** logic expression components like "{PD_SUBSYS1 != PD_SUBSYS1_on && PD_SUBSYS2 != PD_SUBSYS2_on}". Though the testbench is the actual trigger for a power state to make a transition to another state, the cross-coverage collection procedure depends more on unrolling the dependency of these transitions on power states from hierarchical paths. **Example 5** shows a sample report of cross-coverage.

**Example 5** Sample Report for Cross-Coverage based on Example 4.

```
----------------------------------------------------------------------------------------------
UPF OBJECT                                                Metric       Goal    Status
----------------------------------------------------------------------------------------------
 TYPE : POWER STATE CROSS
/alu_tester/dut/PD_SYS(ID:PD1),
/alu_tester/dut/PD_SUBSYS2(ID:PD2),
/alu_tester/dut/PD_SUBSYS1(ID:PD3),
/alu_tester/dut/PD_OUT2(ID:PD4),
/alu_tester/dut/PD_OUT(ID:PD5)
                                                        100.00%        100    Covered
 POWER STATE CROSS coverage instance
    \/alu_tester/dut/pa_coverageinfo/PD_SYS/PD_SYS_PS_CROSS/PS_CROSS_PD_SYS
                                                        100.00%        100    Covered
    Power State Cross                                   100.00%        100    Covered
        bin \PD1:SLEEP-PD2:PD_SUBSYS2_off                     2          1    Covered
        bin \PD1:RUN-PD2:PD_SUBSYS2_on-PD3:PD_SUBSYS1_on-PD4:PD_OUT_on-PD5:PD_OUT_on
                                                              2          1    Covered
```

*G. Case Study: Coverage Computation for Power State Transitions*

The UPF **describe_state_transition** or **add_state_transition** commands, which monitor the legality of power state transitions, are the key for decorating UPF cover-bins for power state transition coverage and provide fine-grain controllability for collecting power state-transition-only coverage. Recalling the semantics in [2], the coverage analytical tool provides the mechanism to include the following coverage information through the **-object** <object_name> options in **describe_state_transition** or [**-supply** | **-domain** | **-group** | **-model** | **-instance**] <object_name> options in **add_state_transition** command. However, any state marked as **-illegal** or state beyond –**complete** will not be covered.

The following examples show the UPF code for power switch power state transition cover-bin decoration mechanisms through the **add_state_transition** command. The tool may generate a coverage report only for the specified transition in <object_name> as shown in **Example 6**.

**Example 6** Controlling Transition Coverage by UPF **add_state_transition** Command.

```
# PSW example for Collecting State Transition Coverage
create_power_switch IN_sw \
      -domain PD_SUBSYS2 \
      -output_supply_port {vout_p VDD_IN_net} \
      -input_supply_port {vin_p MAIN_PWR_moderate} \
      -control_port {ctrl_p IN_PWR} \
      -on_state {normal_working vin_p {ctrl_p}} \
      -off_state {off_state {!ctrl_p}}
# controlling State Transition Coverage by UPF for PSW (IN_sw) shown above
add_state_transition -model IN_sw \
   -transition {t0 -from {ON} -to {}} \
   -transition {t1 -from {ON} -to {OFF}} \
   -transition {t2 -from {ON} -to {}} \
   -transition {t3 -from {ON} -to {ERROR} -illegal}
```

A sample report for the above PSW power state transitions are shown below.

**Example 7** Sample Power state Transition Coverage Reports from **add_state_transition** for PSW.

```
-------------------------------------------------------------------------------------
   UPF OBJECT                                                Metric       Goal    Status
-------------------------------------------------------------------------------------
     TYPE: Power Switch /cpu_tester/dut/IN_sw               50.00%        100    Uncovered
     Power                   Switch                    coverage                   instance
\/cpu_tester/dut/pa_coverageinfo/IN_sw/IN_sw_PS/PS_IN_sw
                                                          50.00%                    100
Uncovered
        Power State ON                                   100.00%        100    Covered
            bin ACTIVE                                         4          1    Covered
        Power State OFF                                  100.00%        100    Covered
            bin ACTIVE                                         2          1    Covered
        Power State PARTIAL_ON                             0.00%        100    ZERO
            bin ACTIVE                                         0          1    ZERO
        Power State ERROR                                  0.00%        100    ZERO
            bin ACTIVE                                         0          1    ZERO
     TYPE: Power Switch /cpu_tester/dut/IN_sw              0.00%        100    ZERO
     Power                   Switch                    coverage                   instance
\/cpu_tester/dut/pa_coverageinfo/IN_sw/IN_sw_PS/PS_TRANS_IN_sw
                                                          0.00%        100    ZERO
        Power State Transitions                           0.00%        100    ZERO
            illegal_bin ON -> ERROR                            0               ZERO
            illegal_bin ON -> PARTIAL_ON                       0               ZERO
            illegal_bin ON -> OFF                              2               Occurred
            bin dummy                                          0          1    ZERO
     TYPE: Power Switch Control Port /cpu_tester/dut/IN_sw/ctrl_p
                                                          50.00%                    100
Uncovered
     Power         Switch         Control         Port          coverage          instance
\/cpu_tester/dut/pa_coverageinfo/IN_sw/ctrl_p/PS_ctrl_p
                                                          50.00%                    100
Uncovered
        Power State ACTIVE_LEVEL                         100.00%        100    Covered
            bin ACTIVE                                         4          1    Covered
        Power State INACTIVE                             100.00%        100    Covered
            bin ACTIVE                                         2          1    Covered
        Power State ACTIVE_Z                               0.00%        100    ZERO
            bin ACTIVE                                         0          1    ZERO
        Power State ACTIVE_X                               0.00%        100    ZERO
            bin ACTIVE                                         0          1    ZERO
     Power         Switch         Control         Port          coverage          instance
\/cpu_tester/dut/pa_coverageinfo/IN_sw/ctrl_p/PS_TRANS_ctrl_p
                                                         100.00%        100    Covered
        Power State Transitions                          100.00%        100    Covered
            bin HIGH_TO_LOW                                    2          1    Covered
            bin LOW_TO_HIGH                                    2          1    Covered
```

It is also worth noting that, during LP-SIM, unlike power states in **add_power_state**, all of the power states originating from **add_port_state**, **add_pst_state**, **add_power_state** may not remain active all the time. In general these commands specify predefined and named power states explicitly for UPF objects; such as power domain, supply set, supply port, and power state tables. The named state and their transitions allow the UPF cover-bin to be automatically computed by the coverage analytical tool. However, in a particular verification environment, the UPF intent file may not specify all possible power states for all of the above objects. Hence the tool specifies the missing power states as "undefined" power states and assigns these states as active when the named states are inactive. Hence the coverage data may also include "undefined" power states and transition information in the coverage data. However, there are always coverage exclusion mechanisms for unreachable or undefined states.

## III. ADAPTABLE COVERAGE DATABASE

One of the main reason of LP coverage remained a missing piece in the entire DVIF, or specifically in dynamic simulation, because there is no UCIS standard for low power coverage based on UPF intents. Fortunately Questa® LP-SIM allows creating automatic LP coverage testplans (or verification plan) during LP-SIM and internally links the testplan with LP coverage objects. Moreover, it also allows annotating coverage data in UCDB.

Mechanisms within the UCDB format allow testplan scopes and coverage scopes to be tagged so that an association can be made between them. This allows the LP testplan to be annotated with LP coverage data and represent LP coverage results based on queries. Even though it is still an *adhoc* approach because it only processes LP coverage, more specifically UPF cover-bins and *UPF cross-cover-bins*. In the succeeding sections we will demonstrate this adhoc approach of UCDB for LP coverage as a case study and thereafter propose a versatile coverage database to adapt both LP and non-LP coverage under the same hood.

### *H. Case Study: Adhoc Approach for LP Coverage Database*

As discussed in previous sections, the LP coverage information created from **List 7** (which are: 1. LP dynamic checks, 2. power states and power state-transitions, and 3. cross-power-domain power-states-dependency-based coverage) provides an extra level of LP verification confidence in the DVIF. Apart from providing the coverage closure metric, their coverage information also facilitates verification productivity by auto generating a LP testplan as a constructive derivative in the process as discussed above. The *Autotestplan* is generated on the basis of coverage information from the UPF cover-bins and *UPF cross-cover-bins* decoration or coverage collection procedure discussed so far. **Figure 5** explains the procedure of generating an LP-testplan.

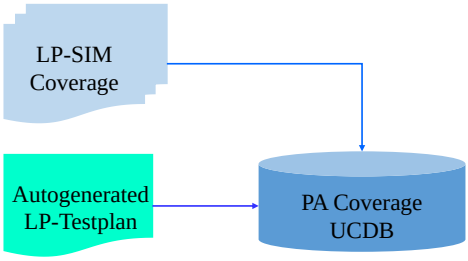


**Figure 5 Autotestplan generation during LP-SIM**

During the regular LP-SIM, the UCDB data are created with "*coverage save -pa pa.ucdb"* tool command. The LP testplan is generated with "*pa autotestplan"*, which will create the data file for UCDB; i.e., "*QuestaPowerAwareTestPlan.ucdb"*. Finally, the LP coverage and testplan coverage can be merged in UCDB to represent LP coverage results as shown in **Figure 6 (a) and (b)**.

| # | Section | Description | Link | Type | Weight | Goal |
|---|---|---|---|---|---|---|
| 1 | cpu_tester | Design hierarchical scope | | | 1 | 100 |
| 1.1 | dut | Design hierarchical scope | | | 1 | 100 |
| 1.1.1 | L1_RET_sw | Power Switch | | | 1 | 100 |
| 1.1.1.1 | Power State Coverage | To cover Power States | L1_RET_sw_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_RET_sw/L1_RET_sw_PS/PS_L1_R | COVERGROUP | 1 | 100 |
| 1.1.1.2 | Power State Transition | To cover Power State Transitions | L1_RET_sw_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_RET_sw/L1_RET_sw_PS/PS | COVERGROUP | 1 | 100 |
| 1.1.1.3 | ctrl_p | Control Port | | | 1 | 100 |
| 1.1.1.3.1 | Power State Coverage | To cover Power States | ctrl_p_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_RET_sw/ctrl_p/PS_ctrl_p | COVERGROUP | 1 | 100 |
| 1.1.1.3.2 | Power State Transition | To cover Power State Transitions | ctrl_p_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_RET_sw/ctrl_p/PS_TRANS_ctrl_p | COVERGROUP | 1 | 100 |
| 1.1.2 | L1_sw | Power Switch | | | 1 | 100 |
| 1.1.2.1 | Power State Coverage | To cover Power States | L1_sw_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_sw/L1_sw_PS/PS_L1_sw | COVERGROUP | 1 | 100 |
| 1.1.2.2 | Power State Transition | To cover Power State Transitions | L1_sw_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_sw/L1_sw_PS/PS_TRANS_L1_sw | COVERGROUP | 1 | 100 |
| 1.1.2.3 | ctrl_p | Control Port | | | 1 | 100 |
| 1.1.2.3.1 | Power State Coverage | To cover Power States | ctrl_p_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_sw/ctrl_p/PS_ctrl_p | COVERGROUP | 1 | 100 |
| 1.1.2.3.2 | Power State Transition | To cover Power State Transitions | ctrl_p_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/L1_sw/ctrl_p/PS_TRANS_ctrl_p | COVERGROUP | 1 | 100 |
| 1.1.3 | CPU_sw | Power Switch | | | 1 | 100 |
| 1.1.3.1 | Power State Coverage | To cover Power States | CPU_sw_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/CPU_sw_PS/PS_CPU_sw | COVERGROUP | 1 | 100 |
| 1.1.3.2 | Power State Transition | To cover Power State Transitions | CPU_sw_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/CPU_sw_PS/PS_TRANS_ | COVERGROUP | 1 | 100 |
| 1.1.3.3 | ctrl_high | Control Port | | | 1 | 100 |
| 1.1.3.3.1 | Power State Coverage | To cover Power States | ctrl_high_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/ctrl_high/PS_ctrl_high | COVERGROUP | 1 | 100 |
| 1.1.3.3.2 | Power State Transition | To cover Power State Transitions | ctrl_high_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/ctrl_high/PS_TRANS_ctrl_hi | COVERGROUP | 1 | 100 |
| 1.1.3.4 | ctrl_moderate | Control Port | | | 1 | 100 |
| 1.1.3.4.1 | Power State Coverage | To cover Power States | ctrl_moderate_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/ctrl_moderate/PS_ctrl_moder | COVERGROUP | 1 | 100 |
| 1.1.3.4.2 | Power State Transition | To cover Power State Transitions | ctrl_moderate_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/ctrl_moderate/PS_TRA | COVERGROUP | 1 | 100 |
| 1.1.3.5 | ctrl_low | Control Port | | | 1 | 100 |
| 1.1.3.5.1 | Power State Coverage | To cover Power States | ctrl_low_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/ctrl_low/PS_ctrl_low | COVERGROUP | 1 | 100 |
| 1.1.3.5.2 | Power State Transition | To cover Power State Transitions | ctrl_low_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/CPU_sw/ctrl_low/PS_TRANS_ctrl_low | COVERGROUP | 1 | 100 |
| 1.1.4 | IN_ISO_sw | Power Switch | | | 1 | 100 |
| 1.1.4.1 | Power State Coverage | To cover Power States | IN_ISO_sw_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/IN_ISO_sw/IN_ISO_sw_PS/PS_IN_ISO_ | COVERGROUP | 1 | 100 |
| 1.1.4.2 | Power State Transition | To cover Power State Transitions | IN_ISO_sw_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/IN_ISO_sw/IN_ISO_sw_PS/PS_T | COVERGROUP | 1 | 100 |
| 1.1.4.3 | ctrl_p | Control Port | | | 1 | 100 |
| 1.1.4.3.1 | Power State Coverage | To cover Power States | ctrl_p_STATE_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/IN_ISO_sw/ctrl_p/PS_ctrl_p | COVERGROUP | 1 | 100 |
| 1.1.4.3.2 | Power State Transition | To cover Power State Transitions | ctrl_p_TRANSITION_COVERAGE,\/cpu_tester/dut/pa_coverageinfo/IN_ISO_sw/ctrl_p/PS_TRANS_ctrl_p | COVERGROUP | 1 | 100 |
| 1.1.5 | IN_sw | Power Switch | | | 1 | 100 |

**Figure 6(a) Autogenerated LP-Testplan**

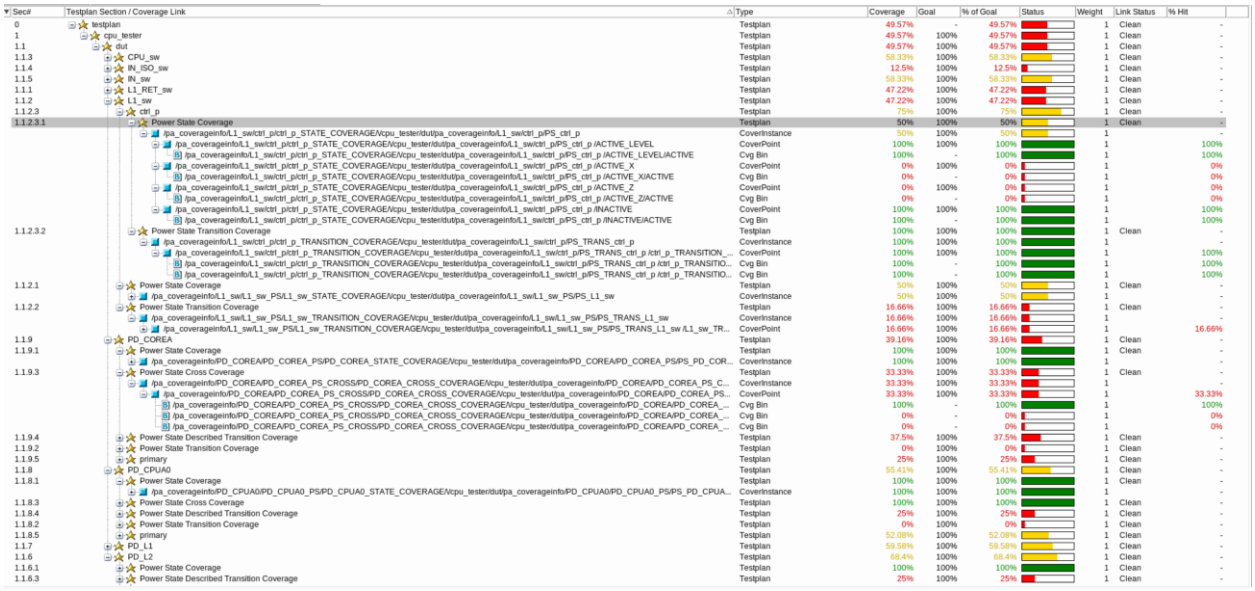


**Figure 6(b) LP-Testplan and LP-Coverage Association**

The highlighted sections "1.1.2.3.1" in **Figure 6(a) and 6(b)** explains how the testplan scopes and coverage scopes are tagged within UCDB to associate the testplan to generate coverinstance, coverpoints and covergroup bins, which are in fact the *UPF cover-bins* and *cross-cover-bins* in the new proposed environment. Almost every coverage object within the UCDB can be linked to the testplan. The UCDB's tagging mechanism allows testplan scopes and coverage scopes tagged within the database. Categorically it allows the association of testplan and coverage objects when a testplan scope and a coverage scope shares the same tag. This allows a consolidated coverage to be calculated and queried based upon the testplan.

### *I. Proposed Adaptive Coverage Database*

While considering a comprehensive and combined database that will coordinate both the LP and non-LP coverage, we emphasized on few aspects like the database must be standard in terms of accessibility and adaptability. The recent version of IEEE-1801-2015 or UPF 3.0 provides a concrete model of LP information database that consists of power-management objects, which are the results of applying UPF semantics

augmentation on HDL designs. Hence the database defines relationship between the design and LP objects, i.e. between the HDL and UPF.

The LP information model database (IMDB) is consists of an LP objects and various information bearing properties defined for those objects. It provides a set of well-defined APIs to access and query the low power or design information in TCL or in HDL. It also offers users the liberty to develop more complex APIs on the basis of fundamental API definitions. The key components of IMBD as discussed so far and defined by the UPF 3.0 LRM are summarized below. To note, even though IMDB is based on UPF 3.0, it remains compatible with all previous versions of UPF, as long the accessibility through API to the model is implicated with HDL.

**List 8:** Components of IMDB

Objects: Are primary holders of information

- They are accessed by handle ID / UPF Handle
- Objects represent UPF, HDL or a relationship between them
- So, there are three major classes of objects
  HDL Objects: Models objects that are representing HDL design
  UPF Objects: Models objects that are created by UPF
  Relationship Objects: Objects that model the relationship between UPF and HDL objects.

Properties: Are collection of information about an object

- They are accessed by property IDs
- Properties are classified into
  Basic Types: String, Integer, Boolean etc.
  Complex Types: Handle to properties, list of handles to other objects etc.
  Dynamic properties: Accessible only from the HDL package functions

**Figure 7** shows the concept of an LP information model database (IMDB).

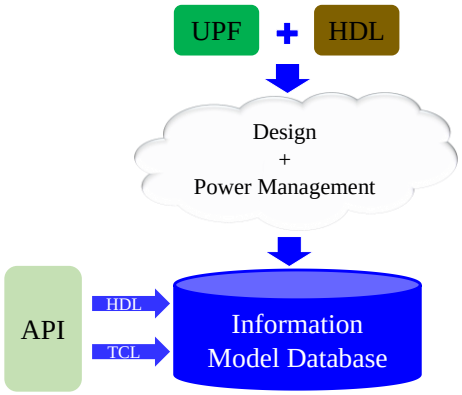


**Figure 7 LP Information Model Database (IMDB).**

Hence, the IMDB is ideal for extending beyond the low power paradigm. Specifically IMDB can be extended to match UCIS standards to analyze, merge and represent LP coverage data on non-LP coverage results. Firstly, because the UCIS API can be used to implement the heterogeneous types of a coverage merge [8], which basically allows coverage analysis tools to merge coverage data from semantically different verification sources, like LP and non-LP sources. UCIS defines that semantically different data need to be stored within the UCDB, but they are usually not automatically merged within the database. If a coverage analysis tool has an algorithm for merging heterogeneous data, it should submit a new data item with a semantic tag indicating merged data.

Furthermore, the UCIS standard also implements an HDL specific unique ID to query through HDL objects for manipulating and representing HDL code coverage related information. Specifically the predefined fields of UCIS composite objects [8] consist of design scopes, coveritems and history nodes that allow relating UCDB coverage data through design HDL objects.

The proposed adaptive coverage database will impose IMDB components (HDL, UPF and relational objects and relevant properties) as a subset of the regular UCIS standard. As a result, we will be able to process both *UPF cover-bins*, *UPF cross-cover-bins* as well as all the non-LP or regular cover bins under the same hood. The coverage data collection, analysis and representation would be possible even from the IMDB as we extend the IMDB HDL components to map the relevant UCIS HDL standard components. The next section explains the provisions for merging LP and non-LP coverage to provide a consolidated coverage metric from all sources and phases of DVIF.

### *J. Algorithm for Merging LP and Non-LP Coverage*

In this paper we have taken the novel initiatives to fulfill the missing pieces of the entire design verification flow — the LP coverage, its standardization and integration with other existing standards for regular non-LP coverage computation models. The methodological approach that we have taken so far can be summarized as follows:

**List 9:** Initiatives at a Glance
- Identify the missing pieces of LP coverage modeling
- Identify the complete source of LP coverage contributors
- Define LP cover bins — the *UPF cover-bins* and *UPF cross-cover-bins*
- Identify LP coverage and testplan association mechanism through UCDB
- Implement standardization mechanism for LP coverage bins through IMDB defined by UPF 3.0
- Extend LP cover bins in IMDB as subset of UCDB
- Identify database accessibility through mapping HDL API defined by both UPF 3.0 and UCDB standards
- Propose adaptive coverage database through UPF 3.0 in IMDB and extend it with UCDB standard for integrating the non-LP coverage,

And finally,
- Identify the requirements of heterogeneous merge algorithms for merging LP and non-LP data in UCDB

So here at the final stage, the algorithm for merging LP and non-LP coverage under extended IMDB (UCDB) is possible through a master testplan as shown in **Figure 7**. A master testplan is a non-LP verification plan generated manually based on the verification specification for the same design in terms of hierarchical structure and scopes that may be used in both LP and non-LP verification flows. The autogenerated LP testplan and master testplan are linked through the coverage objects or test data records to corresponding sections of each other (e.g., refer to **Figure 6(a)** to understand testplan section "1.2.1.3").

Once the links between both the testplans are established under IMDB (UCDB), merging of LP with non-LP *cover-bins* becomes very procedural, since we already know that UCDB allows testplan scopes and coverage scopes to be tagged so that an association can be made between them.

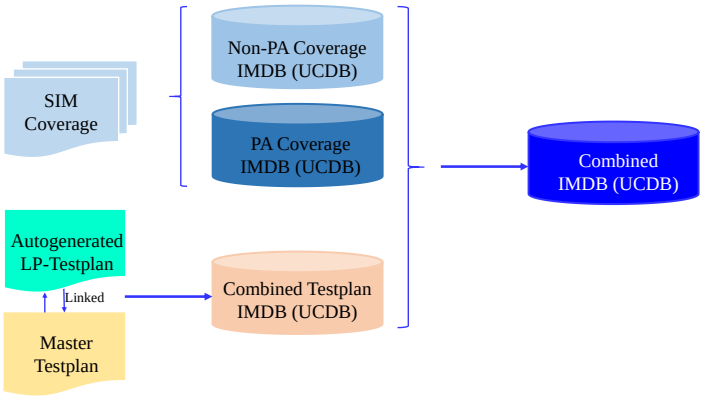


**Figure 8 Heterogeneous Merging of LP and non-LP Coverage in IMDB (extension of UCDB)**

## IV. Future Research and Concluding Remarks

In this paper, we have completed the initial framework for the LP coverage standardization and integration with existing UCIS coverage standards. Further research is required to completely map the functionalities of HDL API defined by both UPF 3.0 and UCDB standards. In addition, further research is also required on the formalization process on what we have mentioned in Section II-D. For a complete RTL code coverage, it is required to make sure that the RTL code coverage data models specifically branch, statement, blocks, condition, expressions etc. for both UPF instrumented RTL and non-LP RTL should be formally or logically equivalent. It requires formal tool to understand the consolidated database like LP and non-LP extended IMDB and coordinate with UPF semantics to conduct equivalency checks at the design circuit level.

## References


[1] P. Khondkar, “Low-Power Design and Power-Aware Verification”, Springer, October, 2017.

[2] P. Khondkar, P. Yeung, et al., “Free Yourself from the Tyranny of Power State Tables with Incrementally Refinable UPF”, February-March, DVCon 2017.

[3] Design Automation Standards Committee of the IEEE Computer Society, “IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems”, IEEE Std 1801-2015, 5 December 2015.

[4] S, Wei Tu, T. Lin, A. Feng, C.Y. Ping, “UPF Code Coverage and Corresponding Power Domain Hierarchical Tree for Debugging”, DVCon 2015.

[5] V. Vikram, S. Awashesh K, “Cross Coverage of Power states”, DVCon 2016.

[6] P. Khondkar, P. Yeung, D. Prasad, G. Chidolue, M. Bhargava, “Crafting Power Aware Coverage: Verification Closure with UPF IEEE 1801”, Journal of VLSI Design and Verification, pp.6-17, Vol. 1, November 2017.

[7] P. Khondkar and M. Bhargava, "The Fundamental Power states: The Core of UPF Modeling and Power Aware Verification", Whitepaper at mentor.com, December 2016.

[8] Accellera System Initiatives, “Unified Coverage Interoperability Standard (UCIS)”, Version 1.0 June 2, 2012.