Unveil the Mystery of Code Coverage in Low-Power Designs: Achieving Power Aware Verification Closure

Madhur Bhargava (madhur_bhargava@mentor.com), Mentor, A Siemens Business
Durgesh Prasad (durgesh_prasad@mentor.com), Mentor, A Siemens Business
Pavan Rangudu (rangudu_pavan@mentor.com), Mentor, A Siemens Business

***Abstract*- With continued advancement in the technology, low-power designs and their verification are becoming more complex. Today's low-power designs make use of various coding styles and have complex power aware and non-power aware macro models. These designs have different flavors of RTL achieved via ifdef macros and run in both low-power as well as functional simulation. The designs also make use of macro models that have power supplies and low-power semantics built in to the design, thus avoiding the need to impart any power aware semantics by the verification tools. Typically, a plan to closure approach for low-power verification requires two things: first, the coverage of power objects (all possible states and transitions) and second, making sure that code coverage of user RTL is complete. Generally, in a low-power simulation, the power aware cells, and the logic to mimic the power behavior, are added to the design RTL itself and, as a result, the user's original RTL gets distorted. In this paper, we will discuss all the challenges in code coverage of low-power designs and also discuss various approaches to overcome those challenges in low-power design code coverage. We will also cover how total coverage results can be visualized in order to achieve verification closure in significantly less time.**

Keywords - Power Management, Power Aware Verification, Low-Power Coverage Methodology

I. Introduction

With the advancement in the technology, low-power designs and their verification is becoming more complex. Today's low-power designs make use of various coding styles and have complex power aware and non-power aware macro models. These chips have multiple power domains, each having multiple operating power modes and dynamically changing voltage levels. The power architecture is controlled with help of a power management unit that issues proper control sequences to different elements of the power architecture. These designs have different flavors of RTL code compiled with ifdef macros and runs in both non low-power simulation and low-power simulation. For the sake of this paper we will refer to the non-power aware simulation as non-pa simulation. The designs also make use of macro models, which have power supplies and low-power semantics built in to the designs, avoiding the need to impart any power aware semantics by the verification tools.

Typically in a low-power design simulation, the power aware cells and the logic to control the power is inserted into the design RTL; as a result the RTL that gets simulated is no longer the same code that the user had written. In order to have the complete verification of the design ensuring that the functionality of the design is intact, it is important to test that all the test vectors are generated, which covers all the code written by the user. It is important to ensure that all the proper test vectors are generated by this unit to verify all power elements and it is even more important to verify the complex interactions between these elements at a higher abstraction level. A plan to approaching closure for low-power verification requires two things: first, the coverage of power objects (all possible states and transitions) and second, making sure that code coverage of user RTL is complete. However power-aware coverage closure is hard and complex in nature.

It is important to note that there is no standardized methodology for coverage of low-power designs as the UPF LRM or the UCIS standard doesn't provide anything to address this. With the help of current UPF standards, verification engineers have taken an ad-hoc approach, along with relying on the verification tools, to achieve coverage of low-power objects, however the RTL code coverage in a low-power design is still a difficult and ambiguous task. This whole process of low-power coverage is tedious and dependent on verification tools and EDA vendors. It is error prone and highly time consuming.

To keep pace with this methodology and address this complex issue of code coverage of low-power designs, we need to first fully understand the flavors of low-power design modeling and the challenges in code coverage of these designs. We will also propose how these challenges are addressed to achieve the results of code coverage. The paper will also include how both the coverage of low-power objects and code coverage can be visualized and contribute equally towards verification closure of the design in a more efficient way thereby saving verification effort and time.

A. Power Intent Specification and Basic Concepts of UPF

IEEE Std 1801™-2015 Unified Power Format (UPF) allows designers to specify the power intent of the design. It is based on Tcl and provides concepts and commands that are necessary to describe the power management requirements for IPs or complete SoCs. A power intent specification in UPF is used throughout the design flow; however it may be refined at various steps in the design cycle. Some of the important concepts and terminology used in power intent specification are the following:

- Corruption semantics: The rules defining the behavior of logic in response to reduction or disconnection of power to that logic.

- Hard macro: A block that has been completely implemented and can be used as it is in other blocks. This can be modeled by a hardware description language (HDL) module for verification or as a library cell for implementation.

- Soft macro: An instance that is represented by the original register transfer level (RTL) and Unified Power Format (UPF) from which its implementation is (or will be) derived. Additionally, ancestor power intent objects are not available for use within the scope of the instance.

- Power state: The state of a supply net, supply port, supply set, or power domain. It is an abstract representation of the voltage and current characteristics of a power supply, and also an abstract representation of the operating mode of the elements of a power domain or of a module instance (e.g., on, off, sleep).

B. Basic Concepts of Code Coverage

Coverage is a metric to assess the progress of functional verification activity. It plays a major role to get a clear picture on how well the design has been verified and also to identify the uncovered areas in verification.
Code coverage and functional coverage are the two types of coverage methods used in functional verification.

Code coverage is a basic coverage type which is collected automatically by the verification tool. It tells you how well your HDL code has been exercised by your test bench. In other words, how thoroughly the design has been executed by the simulator using the tests used in the regression.

Functional coverage measures how well the functionality of the design has been covered by your test bench. In functional coverage the user has to define the functionality to be measured through covergroup and assertion constructs. Functional coverage performed using covergroups or assertions does not have much impact due to low-power semantics and can be achieved in low-power designs easily.

There are different categories of code coverage.

- **Statement Coverage /Line Coverage:** This gives an indication of how many statements (lines) are covered in the simulation, by excluding lines like *module, endmodule, comments, timescale* etc. This is important in all kinds of designs and has to be 100% for verification closure. Statement coverage includes procedural statements, continuous assignment statements, conditional statements and Branches for conditional statements.

- **Conditional/Expression Coverage:** This gives an indication how well variables and expressions (with logical operators) in conditional statements are evaluated. Conditional coverage is the ratio of number of cases evaluated to the total number of cases present.

- **Branch/Decision Coverage:** In Branch coverage or Decision coverage reports, conditions like if-else, case and the ternary operator (? :) statements are evaluated in both true and false cases.

- **Toggle Coverage:** Toggle coverage gives a report about how many times signals and ports are toggled during a simulation run. It also measures activity in the design, such as unused signals or signals that remain constant.

- **State/FSM Coverage:** FSM coverage reports show whether the simulation run could reach all of the states and cover all possible transitions or arcs in a given state machine. This is a complex coverage type, as it works on behavior of the design that means it interprets the synthesis semantics of the HDL design and monitors the coverage of the FSM representation of control logic blocks.

II. Achieving 100% Code Coverage in Low-Power Designs

A few years back users used to have a non-pa version of the design and a low-power version of the same design. The regression setup was also different for both of these versions. For complete coverage, users relied on code coverage of the non-pa version of the design combined with power object's coverage of low-power version of the design. As the technology has advanced, Low-Power is now de-facto in the industry and all the designs are power-aware. So the code-coverage metric needs to be collected on the low-power design itself.

A regression setup is still divided into two parts; the first being a non-pa regression that focuses on verifying the non-pa functionality of the design as if everything is always on. The other is a low-power regression that focuses on verifying the low-power functionality of the design. Typically in a low-power design, a part of the RTL code is covered by the non-pa regression and the remaining part of the RTL code is covered only when low-power regressions are completed. Together they achieve 100% structural code coverage of the designs.

Moreover, to test the design code, it is run with certain testcases just checking the non-pa functionality of the design and some testcases specific to check the power functionality of the design. Together these all testcases contribute towards the coverage of the design. Only if we consider all the testcases we will be able to achieve 100% code coverage.

A. Low-Power Design Modeling

Typically the RTL model of low-power SOCs consists of functional blocks (IPs) modeling the functionality of the design, it also integrates a variety of multi-rail hard macros like IO pads which propagate the supply and signals from the analog to digital world. Together with soft macros and hard macros in the design, a low-power design has a power controller unit. The test vectors for voltage controlling logic are modeled in the testbench. The simulation of SOC having hard macros at RTL relies on simulation models that describe the behavior. These models are provided by IP providers and are developed in HDLs like Verilog or SystemVerilog as described in [4].

To enumerate, a low-power design mainly contains following constructs:

- Soft Macros
- Hard Macros
- Testbench
- Checker/Coverage Modules
- Power Controlling logic

When doing power-modeling of the designs that have hard macros, it is recommended to use Power-Aware behavior models that have a power management interface and power management behavior because code coverage of these models would not be impacted by power artifacts introduced by UPF.

III. Challenges in Code Coverage of Low-Power Instrumented Design

In low-power simulation RTL code is passed to the tool along with a UPF file, the tool inserts the isolation, level-shifter cells and other PA cells into the design. The tool may insert some power logic into the design in order to do power aware simulation. As a result, the original RTL may not be the same as viewed in the source file. The various challenges in low power code-coverage are as follows:

A. Controlling PA logic inside ifdef

In some designs the power aware logic is modeled inside a block with "ifdef PA". That is, the block gets activated only when low-power simulations are run. This code will be covered only when running PA regressions.

Therefore, a code-coverage merge from a low-power simulation and non-pa simulation is required for coverage closure but the merge would not be straightforward as the block is visible in the low-power simulation only.

Coding Guideline

- It is suggested to the designers to use the ifdef PA method to guard the low-power functionality and avoid enabling the PA code in Non-PA runs.

B. Functional Coverage

Functional coverage measures how well the functionality of the design has been covered by your test bench. In functional coverage the user has to define the functionality to be measured through coverage. There are two parts of functional coverage:

- Coverage of assertions (checker logic): In a low-power simulation run, it can happen that during the power off, the assertions can get triggered. These are false alarms and the user expects them to be disabled during the power down period. Simulation tools generally disable assertions during the power down period and also disable the coverage of these assertions during the power down period.
- Covergroup based coverage: Functional coverage done using covergroups does not have any impact and can be easily achieved in low-power designs.

C. Power Controlling Logic: PA Coverage

In a low-power design, it is important to ensure that the complete test vectors are generated by this unit to verify all power elements and it is even more important to verify the complex interactions between these elements at a higher abstraction level. A plan to approaching closure for low-power verification requires coverage of power objects (all possible states and transitions). The low-power coverage is handled separately by the verification tools or it can be modeled in the design as explained in [2].

D. Challenges in coverage of Low-Power Designs having "Hard Macros: PA Behavior Model"

Nowadays low-power designs make heavy usage of Hard Macros, these are power-aware behavioral model in which the power behavior is modeled inside the model itself and is visible only when the UPF connections are made. The PA behavioral model has the following characteristics

- The power pins are not present in the portlist, however declared as wire
- The input power supplies are declared as registers and are initialized to their definitive logical values.

Example

```
module ana_mac(… ip1, ..)
…
wire vdd = 1;
wire vss = 0;
always @(vdd, vss, clk)
  begin
   if (vdd === 1'b1 && vss === 1'b0)
     d = clk & a1;
   else
     d = 1'bx;
  end
```

UPF connections are made to the vdd and vss supply pins in the design. When doing simulation of such models, the tool just makes UPF connections to the supply pins vdd and vss and does not impart any PA logic inside the model.

For code coverage of such models in a low-power design, following items have to be taken into account

- RTL code coverage does not pose any challenge as it is just normal user RTL. The RTL does not get modified by the tool as no PA logic instrumentation happens inside these blocks.
- Toggle coverage of supply pins (vdd, vss) is not considered because these are covered in PA coverage.

- The thing to note here is that code coverage numbers RTL of the hard macro differs in a Non PA simulation as compared to PA simulation run. In a Non-PA simulation run, supplies are defaulted to the ON state and the supplies never goes off. This is different from a low-power run, as the supplies are driven from UPF and can turn OFF as well. As seen in the above example, in the non-pa mode, the always block will get triggered whenever clk changes (as vdd, vss are defaulted to FULL_ON and will not toggle at all), however in PA simulation the always block will get triggered multiple times during power on/off activity.

Coding Guideline

- In order to do macro modeling of Hard IPs, the suggested approach would be to use "PA Behavior Model" or "All pins model" as suggested in [4].
- Also it is suggested to use //coverage off pragmas on UPF supplies modeled in RTL as toggle coverage is not expected on these signals.

E. Soft Macros: RTL Modeling of IPs

In a low-power simulation, RTL code is passed to the tool along with a UPF file, where the tool inserts the isolation, level shifter cells and other pa cells into the design. The tool may insert some power logic into the design in order to do power aware simulation. As a result the original RTL gets modified and no longer remains same.

Consider a simple combinatorial logic.

```
always @(*)
  out = in1 & in2;
```

In above case, whenever 'in1' or 'in2' changes, the statement gets hit. However in a low-power simulation, if the power of this part of design is OFF, then this statement will not get triggered; however signal 'out' will get a value 'x'. When the power is enabled the assign statement will get triggered. Moreover, the number of times this statement gets triggered now also depends on power along with 'in1' and 'in2'.

There is no way in RTL code coverage to capture the activity in the block because of new powering logic.

We look further at various types of code coverage and the challenges in them.

Line Coverage

As the new lines get inserted into the design, it is difficult to do code coverage. In order to simulate the code as per power rules, verification tools introduce new statements while inserting some power logic into the design. As the line numbers get modified in the actual design, performing code coverage on this PA instrumented logic will give improper results as expected on a non-pa simulation. Consider following example:

Example 1: D-FlipFLop with additional PA logic

Actual D-FlipFlop RTL logic	**PA Instrumented D-FlipFlop RTL logic**
1. always @(posedge clk, posedge reset, posedge set) begin 2. if (reset) 3. q<=1'b0; 4. else if(set) 5. q<=1'b1; 6. else if(clk) 7. q <= d;	1. always @(posedge clk, posedge reset, posedge PWR) begin 2. if (~(PWR)) 3. q <= 1'hx; 4. else 5. if (reset) 6. q <= 1'h0; 7. else if (set)

8. end	8. q <= 1'h1; 9. else if (clk) 10. q <= d; 11. end

In this example, we observe that new statements/lines introduced into the RTL logic (lines 2, 3, and 4). The insertion of these new statements to govern power logic is inevitable and therefore calculating code coverage on this PA instrumented RTL logic will not give proper results as expected on a non-pa simulation.

Conditional/Expression Coverage

In addition to introducing new lines/statements into RTL logic, some power logic gets inserted into RTL statement in case of expressions. Consider the following example:

Example 2: PA logic inserted into RTL Expression

Actual RTL Expression	PA-Instrumented RTL Expression
assign c = a&b;	assign c = (PWR) ? (a&b) : 1'bx;

In the case of non-pa logic, the input terms for Expression coverage are **a** and **b.** Now, with the introduction of power logic into the RTL expression, the input terms for Expression coverage will be **PWR, a** and **b.** Therefore, with the increase in FEC Expression input terms, coverage results will not give proper results as expected on a non-pa RTL logic.

Similarly, consider the following example:

Example 3: No change in RTL Conditional statement

Actual RTL Conditional statement	PA-Instrumented Conditional statement
1. always @(a, b) begin 2. if (a & b) 3. c = 1'b1; 4. end	1. always @(a,b,PWR) 2. if (~PWR) 3. c = 1'hx; 4. else 5. if (a & b) 6. c = 1'h1;

We can observe that no new term has been introduced in the conditional expression **"if (a & b)"** and therefore, the user can have proper results as expected on a non-pa simulation with respect to condition coverage.

Coding Guideline

- The suggested way to model the above functionality would be the use case as explained in Example 3 as it does not post low-power coverage challenge.

Branch Coverage

In order to insert PA logic verification, tools not only introduce new lines/statements but also new branches into the design (in most cases to specify the power down logic). We can observe the same from **example 1,** mentioned above

where new branches at lines 2 and 9 got introduced to include power logic in the actual RTL logic design. As new branches are inserted into the design, performing code coverage on this PA instrumented logic will include additional coverage elements under Branch coverage and thus users cannot have proper results as expected on a non-pa simulation.

Toggle Coverage

Insertion of low-power logic affects toggle coverage as well. The purpose of toggle coverage is to report how many times signals and ports are toggled during a simulation run. But with the insertion of power logic into the RTL logic, toggling of signals and ports may increase and therefore performing code coverage on this low-power instrumented logic will not give proper results. Consider the following example:

Example 4: Change in Sensitivity list can affect Toggle coverage

Actual RTL logic	**PA Instrumented RTL logic**
1. always @(a) begin 2. t = 1'b1; 3. #1 t = 1'b0; 4. end	1. always @ (a, PWR) 2. if (~(PWR)) 3. t = 1'bx; 4. else 5. begin 6. t = 1'b1; 7. #1 t = 1'b0; 8. End

As we can see, the always block can get triggered whenever there is an activity on **PWR** signal even though there is no activity on signal **a**. Assuming no activity on signal **a,** then toggle coverage for signal **t** will be zero in the case of a non-pa simulation. But due to additional power logic, whenever there is an activity on PWR signal, signal **t** will toggle thus achieving 100% toggle coverage for signal **t**. This is one simple scenario which explains the effect of introducing additional power logic (into RTL logic) on toggle coverage.

State/FSM Coverage

FSM (Finite State Machine) coverage reports identify whether the simulation run could reach all of the states and cover all possible transitions, or arcs, in a given state machine. In a non-pa simulation, the user assumes the design to be powered-up during the entire simulation run and defines various states and conditions required for an object to reach a particular state. However during a low-power simulation, when the power goes off the object should be in some state (other than the states defined by the user) and therefore verification tools add new states inside the state machine say, **xx.** With the introduction of new state, FSM coverage in low-power simulation run will not give proper results. Consider the following example:

Example 5: New state introduced into FSM as a result of PA Instrumentation

Actual FSM	**PA Instrumented FSM Logic**
1. reg [1:0] state; 2. parameter s1 = 2'b00, s2 = 2'b01, s2 = 2'b10, s3 = 2'b11; 3. always @(posedge clk or posedge reset) begin 4. if (reset) 5. state <= s1;	1. reg [1:0] state; 2. parameter s1 = 2'b00, s2 = 2'b01, s2 = 2'b10, s3 = 2'b11; 3. always @(posedge clk, posedge reset, negedge PWR) begin 4. if (~(PWR))

```
6.  else case (state)
7.      s1:if (in == 1'b1) state <= s2;
8.      else state <= s3;
9.      s2: state <= s4;
10.     s3: state <= s4;
11.     s4: state <= s1;
12.     endcase
13.     end
```

```
5.      state <= 2'hx;
6.  else
7.      if (reset)
8.          state <= 2'h0;
9.      else
10.         case (state)
11.             2'h0:
12.                 if (in)
13.                     state <= 2'h1;
14.                 else
15.                     state <= 2'h2;
16.             2'h1:
17.                 state <= 2'h3;
18.             2'h2:
19.                 state <= 2'h3;
20.             2'h3:
21.                 state <= 2'h0;
22.         Endcase
23. end
```

From the low-power instrumented FSM logic, we infer that when the power goes down, the value of the state will be **2'hx.** Since there is no state associated with that value (from the 4 states s1, s2, s3, s4 defined by the user in RTL design), verification tools create a new state (say, **xx**) to make sure the object is in some state during the simulation run. Since FSM coverage reports how many states and transitions in between these states are hit during the simulation run, this introduction of a new state (in the case of low-power Instrumented FSM logic) will not give proper results.

Another challenge would be whether to display coverage results when the design is powered-off since the user hasn't specified any logic in that case. Therefore displaying code coverage for the logic (additionally added as a part of low-power instrumentation) would not make sense from user's point of view.

Thus, we have highlighted some of the major challenges in all types of code coverage like line coverage, branch coverage, expression coverage, toggle coverage and FSM coverage. In the next section, we will discuss various approaches to overcome the challenges in low-power design code coverage.

IV. Addressing the Challenges of Code Coverage in Low-Power Designs

In a low-power simulation due to low-power instrumentation the coverage numbers are expected to be different from non-pa simulation. However we need to address the visualization of coverage results for a low-power design. We will look into various approaches in each of the code coverage types to tackle this issue.

Line Coverage

The issue with Line coverage calculation in a PA simulation run is that including power logic introduces new lines/statements which not only modifies line numbers of existing RTL logic but also gets included in Statement/Line coverage.

One way to handle this problem is to exclude the coverage of new lines/statements introduced by including power logic. In example 1, exclude the statement coverage for statements (line 2, 3, and 4) so that it will give proper results as expected on a non-PA RTL logic. Also, the line numbers of newly added lines can be set to some high number/negative number so that they won't modify the line numbers of actual non PA-RTL logic while reporting.

Conditional/Expression Coverage

Considering **example 2** for low-power instrumented expression, we observe that actual non-pa expression has been modified to include power logic and did not introduce any new lines/statements. This modification resulted in an increase in FEC expression input terms and therefore expression coverage will not give proper results as expected on a non-PA RTL logic.

In this case, one cannot simply apply the above approach (excluding statements introduced for including power logic) because no new line/statement has been introduced here. In order to achieve proper low-power expression coverage one can exclude the input terms that have been additionally added. Once they have been excluded, the expression coverage will depend only on the actual RTL logic input terms and thus give proper results as expected on a non-pa simulation.

From the given example 3, there is no issue with Conditional coverage because the conditional statement has not been modified and thus one can have proper results as expected on a non-pa simulation.

Branch Coverage

The issue with Branch coverage calculation in a low-power simulation run is similar to that with Line coverage calculation. In this scenario,to include power logic new branches are introduced which not only modifies line numbers of existing RTL logic but also gets included in Branch coverage.

Here, again, one can use the same approach followed for Line coverage to get achieve sophisticated code coverage, for example, excluding the coverage of new branches introduced to include power logic. In the given **example 1**, exclude the branch coverage for newly added branches (lines 2 and 9) so that it will give proper results as expected on a non-pa simulation. Also, the line numbers of newly added branches can be set to some high number/negative number so that they won't modify the line numbers of actual RTL logic while reporting.

Toggle Coverage

Insertion of power logic into the RTL logic causes toggling of signals and ports which are otherwise unused signals or signals that remain constant as in non-pa simulation. Therefore performing toggle coverage on this low-power instrumented logic will not give proper results.

As we can see in example 4, the always block can get triggered whenever there is an activity on **PWR** signal even though there is no activity on signal **a**. This triggering of always block will result in toggling activity of signal **t.** Since, there is no way in RTL code coverage to exclude the toggling activity of signals in the block because of new powering logic, there will always be a difference in toggle coverage results in both non-pa and low-power simulation.

State/FSM Coverage

The issue with State/FSM coverage is that introduction of any new states will not give proper results as expected on a non-pa FSM logic. From the low-power instrumented FSM logic in example 5, we infer that verification tools create a new state (say, **xx**) to make sure object is in some state during the simulation run when power goes down.

One way to tackle this issue is to exclude the states added to facilitate the power logic which will result in proper FSM coverage as expected on a non-pa simulation.

Complex PA Instrumentation

Please note that the given examples are simple RTL logics for which we could easily apply the above mentioned approaches to tackle the challenges imposed by additionally added PA instrumented logic and thus achieve proper code coverage results as expected by the user on a non-pa simulation. There may be various design logics which result in complex PA instrumentation for which exclusion of code coverage components for additionally

added PA logic wouldn't be simpler. Consider the following example in which retention strategy has been applied on a simple D-FlipFlop:

Example 6: PA instrumentation of Restore and Save processes on output of a D-FlipFlop

Actual D-FlipFlop RTL logic	**PA Instrumented D-FlipFlop RTL logic with Retention capability**

always @(posedge clk,posedge reset, posedge set) begin
  if (reset)
  q<=1'b0;
  else if(set)
  q<=1'b1;
  else if(clk)
  q <= d;
end
	always @(posedge clk, posedge reset, posedge set, negedge posedge restore_signal_is_active, posedge restore_condition_is_true)
	begin
	 if (~(PWR))
	  q <= 1'hx;
	else if (PWR && RET_PWR && ! (<restore_signal_is_active> && <restore_condition_is_true>))
	begin
	//**original_process_body**
	  if (reset)
	  q <= 1'h0;
	  else if (set)
	  q <= 1'h1;
	  else if (clk)
	  q <= d;
	end
	// **Restore process**
	else if (PWR && RET_PWR && <restore_signal_is_active> && <restore_condition_is_true> && ! (<save_signal_is_activ && <save_condition_is_true>))
	  q <= <retained_value>
	else
	  q <= 1'hx; // <corrupt_value>
	end
	// **Save Process**
	always @(save_signal_is_active, save_condition_is_true)
	begin
	if (RET_PWR && <retention_cond_is_true>
	&& ! (<save_signal_is_active> && <save_condition_is_true>
	begin
	end
	else if (RET_PWR
	&& <save_signal_is_active> && <save_condition_is_true>
	&& ! (<restore_signal_is_active> && <restore_condition_is_
	 <retained_value> <= q // saving register value
	else
	 <retained_value> <= 1'hx; //<corrupt_value>
	end

In this example, in order to provide retention capability, the behavior of each register to be retained is modified. We noticed that the process sensitivity list for the register is expanded and the body of the process is modified. Also, an additional process is created for the saved state. Excluding coverage for new statements/branches introduced into the RTL logic is not straightforward and therefore, addressing this kind of complex PA instrumented scenarios need more diligent and meticulous analysis.

The analysis to divulge the various approaches to exclude coverage for additionally introduced statements/branches is still in its embryonic stage and some more work in future is needed for this. This work hopes to be the first step towards this direction of finding some ways to address the challenges of Code coverage in Low-Power designs.

V. Achieving the Coverage Closure in Low-Power Designs

Code coverage tells us the quality of implemented design only and not the low-power functionality; thus it alone doesn't ensure verification completeness. With 100% coverage in both code coverage and functional coverage one can have verification closure of the design.

Low Power coverage is as essential as functional coverage of RTL. Low-Power coverage data helps to ensure that regression suites are adequately testing power aware elements of the design, including dynamic power aware checks and power aware states and transitions, as created through the UPF file with the add_power_state, add_port_state or add_pst_state commands.

SystemVerilog construct "Covergroups" help in sampling signal/property activities at desired sampling points through coverpoints and bins. These are used effectively to collect coverage numbers of states and their transitions. Moreover, low-power coverage methodology [2] which combines UPF 3.0 HDL functions with SystemVerilog coverage constructs can also help the user to write fast and reliable low power coverage infrastructure.

Power/Port/PST states are a key aspect of today's low power designs. They capture the intent about the operating modes of the low power design and hence have a huge impact on the functionality. So it becomes essential to verify the occurrence of various states and their transitions to ensure proper operation of low power designs as well as to achieve verification closure of the design.

Low-Power coverage metrics:

a) Power/Port/PST State coverage : Ensures that all the states are exercised during verification
b) Power/Port/PST Transition coverage: Ensures that all valid transitions are covered. Moreover any illegal state transition occurrence can be identified.
c) Cross state coverage: Inter-dependent power domains lead to various combinations of power states. These interconnections affect the placement of power management cells. Cross states coverage verifies these combinations.
d) SimMode coverage: Ensures that all specified simstates of supply sets and their transitions are covered during verification.
e) Save/Restore Event coverage: Ensures proper save/restore events per strategy happened at least once during verification.
f) Switch_Port coverage: Ensures that all the states on the control_ports and ack_ports of power switches and their transitions are covered during verification.
g) Isolation_Signal coverage: Ensures the toggling activity of Isolation signal during verification.

Similarly, one can think of other coverage metrics based on different properties of upf objects & requirement and thus, calculate low-power coverage which when combined with code coverage results in verification closure in Low-Power designs.

VI. Conclusion

The complexities in low power verification have greatly increased. Code coverage is an important aspect of verification process. In this paper, we have discussed all the challenges in code coverage of low-power designs and also discussed how these challenges are addressed in low-power verification environment. We have also covered how total coverage results can be visualized in order to achieve verification closure in significantly less time.

VII. References

[1] IEEE Std 1801™-2015 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 05 Dec 2015.

[2] "Awashesh Kumar, Madhur Bhargava", Unleashing the Power of UPF 3.0: An innovative approach for faster and robust Low-power coverage, DVCon India 2017

[3] "Pankaj Kumar Dwivedi, Amit Srivastava, Veeresh Vikram Singh", Let's DisCOVER Power States, DVCon USA 2015

[4] "Mohit Jain, Amit Singh, JSS Bharat, Amit Srivastava, Bharti Jain", Overcoming barriers in Power Aware Simulation, DVCon Europe 2014