Confidently Sign-off any Low-Power Designs without Consequences

Madhur Bhargava (Madhur_bhargava@mentor.com), Siemens EDA
Jitesh Bansal (Jitesh_bansal@mentor.com), Siemens EDA
Progyna Khondkar (Progyna_khondkar@mentor.com), Seimens EDA

Abstract- Successful verification of low-power designs has always been a challenge in the industry. The IEEE 1801 (UPF) standard introduces low-power modeling concepts into today's complex designs. Different empirical studies shows that 40% or more of engineering time-effort is typically spent only on debug for any design verification project. For low-power design and verification, these debug challenges are further complicated, resource and time consuming as a consequences of complicated power management architectures and instrumentations implication on actual design. With increasing SoC complexity, it is inevitable to comprehend the common low-power design issue so that these can be avoided or caught early during the SoC design cycle and save on debug time-effort or even re-spin of the chip. Thus, a well implied low-power verification methodology is important for signing off with confidence. In this paper, we will provide an in-depth analysis of various low-power design issues that are on a faced daily basis. By taking relevant examples and case studies, this paper demonstrates how these issues can be either avoided or solved during RTL bring up phase. Based on the dynamic verification & coverage techniques, this paper includes an efficient low-power verification methodology which can be followed for signing off the chip without consequences.

I. Introduction

Power Management & Verification Complexity

The growing demand for energy efficient electronic systems has resulted in sophisticated power management architectures. The constant need to minimize energy consumption to increase battery life for portable devices, and reduce heat dissipation for non-portable devices to minimize cooling costs ensure that power management is critical part of any electronics designs. Designers employ a variety of advanced techniques ranging from clock gating and power gating to multiple voltages and dynamic scaling of voltages and frequency. If not executed correctly, these techniques can affect the functionality of the design. Hence, it becomes important to verify the power management to ensure functional correctness of designs. It is equally important to catch and rectify the low-power issues as early in the design cycle as possible saving on development cycles and any re-spins.

The application of power management results in the modification of the original design and insertion of special power management structures like isolation, level shifters, retention, etc., at various places in the design. Due to the complex interaction of these structures with the normal design functionality, it poses a serious of challenge to verification. To add to it, the various IPs with their own power management need proper verification to remove any integration issues related to power management. With these ever-increasing complexities, it is highly significant to understand the possible low-power design issues so that these can be avoided or caught early in the design cycle. The paper discusses a low-power verification methodology which can be followed to catch the low-power issues early in the design cycle, that is, at the RTL stage itself. The paper also includes common low-power issues, their probable root causes and debugging tips.

Power Intent Specification and Basic Concepts of UPF

IEEE Std 1801™-2018 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 in this paper are the following:

- Power domain: A collection of HDL module instances and/or library cells that are treated as a group for power management purposes. The instances of a power domain typically, but do not always, share a primary supply set and typically are all in the same power state at a given time. This group of instances is referred to as the extent of a power domain.

- Isolation Cell: An instance that passes logic values during normal mode operation and clamps its output to some specified logic value when a control signal is asserted. It is required when the driving logic supply is switched off while the receiving logic supply is still on.

- Level Shifter: An instance that translates signal values from an input voltage swing to a different output voltage swing.

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

- 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).

- Retention: Enhanced functionality associated with selected sequential elements or a memory such that memory values can be preserved during the power-down state of the primary supplies.

- Retention register: A register that extends the functionality of a sequential element with the ability to retain its memory value during the power-down state.

- Driver supply: For a driver that is a transistor, the supply connected to its source or drain; for a driver that is an inverter, the pair of supplies connected to the source/drain of the transistor pair comprising the inverter; or for an output of an active component, the related supply set of that output.

II. Motivation for Methodology

To achieve comprehensive low-power verification, a verification engineer is interested in a number of questions, such as:

- Is the design free of all common low-power design issues?

- Are enough test vectors specified to cover all possible combinations to catch low-power design issues?

- Design may have some specific scenarios and functionality which is critical for functioning of the chip. In such cases, how can it be ensured that those specific conditions are checked in the design?

In order to address the above questions, it is essential to understand all the possible common low-power design scenarios. There needs to be a straight-forward way to catch all those low-power issues early in the design cycle, debug the failures and do its root cause analysis to fix the issues in the design. It is equally important to ensure that enough test vectors are provided ensuring that low-power assertions are checked properly to catch any probably issue in the design.

Tool generated assertions (or low-power checks) are used widely in low-power verification. However they may not be exhaustive in all the designs, as highlighted by the reasons below:

• A design can have a very specific requirement which is not being provided by the tool-generated low-power checks

• The low-power technology is still evolving and hence a new set of protocol appears every now and then, which may require a different set of checks which is not yet supported by the verification tool.

These requirements can be met through SystemVerilog assertions. The immediate assert construct of SystemVerilog triggers at the activities of a signal and checks for a specific condition to be true. SystemVerilog assertions are fairly common way to check for various design properties. Due to the above reasons the user may want to write custom low-power assertions using the SystemVerilog assert features. These items can be grouped in a checker module, and this checker module can be instantiated into the design using UPF command “bind_checker”.

Having a clear understanding of various low-power design issues and combining the above two approaches, effective and efficient low-power verification can be done.

III. Low-Power Design Issues & Debugging Strategies

Following discussions uphold most common but overwhelming and challenges low-power design issues sub-divided into various categories.

A. *Isolation Issues*

1) Isolated Port Value is different from Clamp Value

If the isolated port value is different from clamp value at the time isolation is enabled/disabled, it can lead to a state change at the time of isolation enable / disable. In this case, the logic reading the port sees a different value at isolation enable/disable since the new value is the clamp_value.

- Root Cause
  - This indicates that either the port is isolated with a wrong strategy (unexpected clamp value) or the inputs are not consistent with the clamp value.
- Debugging Strategy
  - In the dynamic simulation waveforms, check the value of isolated port, isolation control, clamp value of isolation strategy to identify the mismatch.
  - If during the isolated period, the clamp value was found different from isolated value at the port, indicating problem in the instantiated isolation cell. This can also happen if the supply of isolation cell got switched off during isolation period.
  - Another possible reason could be that the isolation control signal is active when there is no requirement for isolation indicating a redundant cell.
  - Isolation control is not enabled when power is switched OFF. This is a protocol issue and can result in serious problems in the design.
  - If Isolation clamp value is same or different from the Reset value

2) Activity on Isolated Port at / during the isolation period

Any activity on the isolated port during the isolation period indicates the port is no longer clamped to a constant specified value. Similarly at the time of enable / disable of isolation the toggling of isolated port indicates a potential problem in the design as the state change can happen.

- Root Cause
  - This indicates a potential problem in the control or supplies of isolation cell or a potential problem in the cell itself if this is already instantiated in the design.
- Debugging Strategy
  - In the dynamic simulation waveform, check the value of isolated port, isolation control, and supplies of isolation cell
  - If the supplies are turning off during the isolation period, the output can be corrupted
  - If the cell is already instantiated in the design, it indicates an issue in the cell itself.

3) Incorrect value on isolated port during Non-Isolation period

During the Non-Isolation period, the output of isolation cell should be same as its input. That is, during the non-isolation period the cell should behave as an AND gate. Any difference can lead to a problem in the chip

- Root Cause
    - This indicates a potential problem in the control or supplies of isolation cell or a potential problem in the cell itself if this is already instantiated in the design.
- Debugging Strategy
    - In the dynamic simulation waveform, check the value of isolated port, isolation control, and supplies of isolation cell
    - If the supplies are turning off during the non-isolation period, the output can be corrupted. This is applicable only in the duration when sink supplies is ON.
    - If the cell is already instantiated in the design, it indicates an issue in the cell itself.

4) Redundant Activity on isolation control signal

Isolation should be enabled by turning on control signal only at the time when source is OFF and sink is ON. Any additional activity isolation control is redundant and indicates an issue in power controller.

- Root Cause
    - This indicates a potential problem in power controller logic
- Debugging Strategy
    - Add to dynamic waveforms, supplies of source & sink logic, isolation control
    - Tracing back on isolation control will help determine the real issue in power controller.

5) Isolation turned off during the power-shut off or COA

Isolation should be enabled at all times when source is either in Corrupt or COA. If isolation is turned off during the time it is required can lead to major issues in the design.

- Root Cause
    - This indicates a problem in power controlling logic which controls the isolation signal
- Debugging Strategy
    - Add to dynamic waveforms, supplies of source & sink logic, isolation control
    - Tracings back on isolation control will help determine the real issue in power controller.

B. Level Shifter Issues

1) Missing level shifters in the design

Whenever there is a logic crossing from a source domain to sink domain with voltage difference between the two, a level-shifter needs to be inserted in the design.

- Root Cause
    - If a signal having a logical value “1” generated from source domain A operating at 3 V goes to sink domain B operating at 5V however its logical value is “x”. This indicates a missing level shifter in the design.
- Debugging Strategy
    - Add to dynamic waveforms, supplies of source & sink logic, signal in question

2) Incorrect level shifter inserted

Whenever there is a logic crossing from a source domain to sink domain with voltage difference between the two, level-shifter needs to be inserted in the design.

- Root Cause
    - In the cases where source domain operates at a higher voltage range compared to sink domain, there is a need for a high-to-low level shifter. Inserting a wrong level shifter can pose serious design issues
- Debugging Strategy
    - Add to waveforms sink & source supplies and check for voltage difference
    - Add to waveform level shifter type

C. Retention Issues

1) Retention protocol issues for Master Slave configuration

There are multiple ways in which designers model retention. One such way is Master slave configuration. In a master/slave-alive retention register, the retained value is held in the master or slave latch. In this case, the retention element is in the functional data-path of the register. The retention behavior will work fine only when the right protocol or sequence of control is followed. Some of the conditions to be maintained are follows -:

- Retention condition should be asserted high when the power is switched off.
- Retention condition should not toggle during power down. Toggling of the retention condition during power down corrupts the retained value.

Failure to not follow the above rules can lead to problems in the design. Debugging such failure requires -:

- Add to dynamic waveforms, retention condition, retained signal, power supply and retention supply.

2) Value mismatch in saved and restored value

Retention is applied on the designs to make sure that we save the value of sequential elements before power off and once the power comes back the same value gets restored. This ensures the right retention operation and design remains in a valid state. If this sequence of values is not met, it is a serious problem in the design.

- Root Cause
    - This can happen because of problem in retention control, its supplies or the cell itself.
- Debugging Strategy
    - Add to waveform retention controls, retention supply, retained signal.
    - If supply of retention goes off anytime during the retention period, the retained value will get corrupted
    - If the cell is already instantiated in the design, such a failure indicates issues in cell itself. That is, retention model specified by user

D. Miscellaneous Low-Power Design Issues

Some of other miscellaneous common low-power design issues are as follows.

- Isolation or retention supplies are switched off during the active isolation or retention period
    - Switching of the supplies of isolation or retention can lead to corruption of signal and a problem in the design. To debug such issues, add to waveform the strategy supplies and controls. During the active isolation or retention period, supplies should be ON.
- Control ports Switch/Iso/Ret strategy got corrupted.
    - If the control signals of any strategy get corrupted, it could lead to strategy not behaving correctly and as a result issue in the design.
    - It is possible that these controls have buffer placed in the path or generated from a domain which is relative less On.

- Spurious spikes (glitches) on control lines can cause false switching of control ports of various control logic (such as isolation, power switch, and retention).
- Undefined or illegal state on a PST/ Supply port/ Power Domain / Supply Set / Power State Group reached.
    - It is important for design to remain in valid state. System going to invalid or illegal state indicates issue in the design.
- Input to a power domain toggled when the power domain is turned off.
    - This is a redundant toggling activity and wastage of power.

IV. Proposed Verification Methodology

As described in the previous sections, there are good number of low-power issues that can crop up in the chip and lead to functional failures. Catching these issues early in the design cycle is of utmost importance to save on critical chip development time and avoid any re-spins. Most of the verification tool offers capabilities in terms of static and dynamic checking to catch low-power issues. Many of the low-power design issues discussed in the previous sections can be caught during dynamic simulation through certain pre-implemented checks in the verification tools. However, with the increasing complexities in the design, the above list of low-power design issues may or may not be complete.

There could be targeted low power checks specific to a custom design requirement. Effective verification with UPF is very important since there is no "one solution that fits all". Hence the proposed methodology also makes use of custom dynamic checking technique using UPF Information model and UPF bind_checker syntax and semantics. Together with simulator's dynamic checks and user defined custom low-power checks using bind_checker, verification engineers can ensure all their common low power requirements are met. So, having the infrastructure to perform automated dynamic checks plays significant roles to capture common, complex and / or custom low power design issues through user bind_checker assertions.

With verification tools dynamic checking capability combined with user's custom low-power assertions, one is ensured that all the low-power checks are built into the design or simulation run. However, one important thing to note is that enough test vectors will be required to capture issues or exercise all possible scenarios. If sufficient test vectors are not provided, all these checks will not be exercised. In such scenario even though error will not be emitted out of simulation output, however the real issues are still present in the chip because the check never got exercised at the first place.

Let's consider the common Isolation issues listed in section (IV), i.e., "Isolation port value should be same as clamp value at the time of isolation enable/disable" and a situation where isolation strategy is applied with incorrect clamp value. Please note that the check will get triggered checking the scenario only at the time of isolation enable/disable. If the isolation is never enabled with isolation signal being constant "0", the check will never get exercised and the issue won't get caught at all.

To avoid such issue and ensuring that all test vectors are met, it is important to do coverage of all low-power checks. If a particular check is not failing in dynamic simulation and still has zero coverage, then verification engineers need to rework on test vectors and further continue with testing. The goal is to ensure that none of the checks (tool's assertions) and user low-power assertions have zero coverage. It is easy to analyze checks coverage if one single snapshot is provided for both tool and user low-power assertions. If any of the checks fail in the design, the output will have an Error / Failure. This error needs to be debugged using the techniques mentioned in previous section and further it needs to be fixed. Once all the errors are fixed, simulation needs to be rerun to ensure there are no further failing low-power assertions. The target is to make sure simulation output is free of errors and coverage of all low-power checks is 100%.

Below is the flow diagram of proposed methodology.

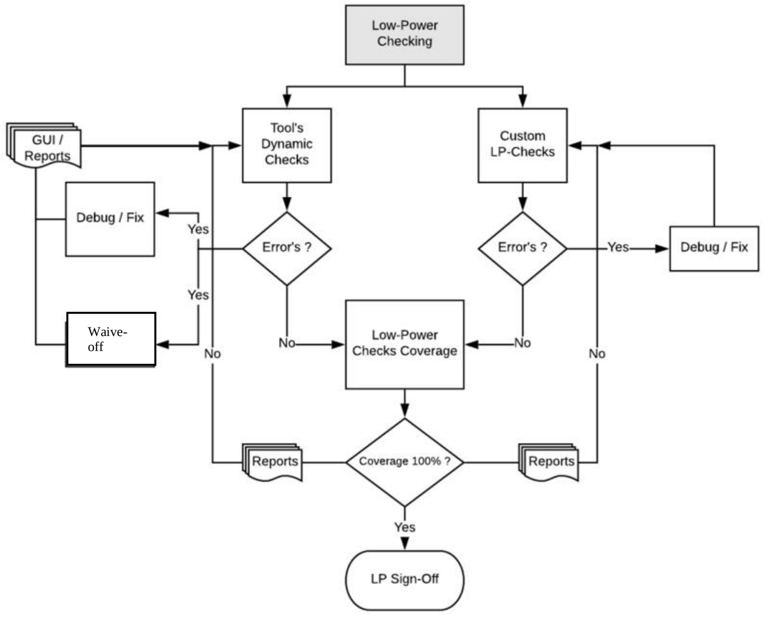

Figure.1 Proposed Verification Methodology Framework

It is possible that the simulation output has failing assertions or errors which are expected as per the design intent. For example, a check can fail stating glitches on the control signal. However, in the real design these glitches may be expected or the glitch window can be more. In those scenarios, the verification tools provide waiver mechanism to waive or exclude such failing checks. In the above example, we can specify our glitch window to be 5 ns and any signal having activity inside that window only will fail. Control signals toggling in the range > 5 ns, will not be considered as glitch. Another example would be when Isolation checks fails because of redundant activity on its control signal. In the design that control may be coming from another design and the activity might be expected. In such cases, user can simply waive off the failing check.

Once there are none failing checks (either fixed or waived off) and the low-power checks coverage is 100%, the low-power sign-off of the chip can be done.

V. Writing Custom Low-Power Checks

One of the important aspects of above low-power methodology is to write custom low-power checks or assertions.

The proposed checks methodology involves getting the handle of low-power object in HDL using the information model and querying the properties of that object. These properties are then further passed to the checker module as port mapping. The checker module has system Verilog assertion to check for a specific property like checking that a combination of state for two different domains never occur simultaneously.

Steps involved in the proposed methodology:

1. Low power object handle: First step is to get the handle of the low power object. This can be achieved by using the HDL package function *"upf_get_handle_by_name"*. A property which is a list can be iterated through the HDL access function *"upf_iter_get_next"*. Following SV code illustrates the usage of these basic function to access low power objects.

```
upfHandleT pd = upf_get_handle_by_name("/tb/dut/pd")
upfHandleT pd_state_list = upf_query_object_properties(pd,
                                                UPF_PD_STATES)
upfHandleT pd_state = upf_iter_get_next(pd_state_list);
```

2. Dynamic property(value) of low power object: Get the dynamic value of the property of interest e.g. user might be interested in knowing the current power state of a power domain and intends to check the state of this power domain w.r.t. the current power state of some other power domain. Also the user may use UPF 3.0 continuous access HDL package function to continually monitor the power state of the power domain and may use assertions for the cases of interest/anomaly.

```
      upfPdSsObjT pd_hdl;

      // This creates a mirror – the value of object created in UPF is 
mirrored to RTL object pd_hdl via IMDB HDL API
      upf_create_object_mirror("/tb/dut/pd", "pd_hdl");

      // Get the handle for current power state of pd
      upfPowerStateObjT pwr_state = pd_hdl.current_state;

      // Query the name of domain and its power state
      upfHandleT pd_name = upf_query_object_properties(pd_hdl.handle, 
UPF_NAME);
      upfHandleT state_name = upf_query_object_properties(pwr_state.handle,
                                                UPF_NAME);

      // At change in value of domain handle, print its state
      always@(pd_hdl)
             $display( "Power domain %s, is in Power State: %s, 
             upf_get_value_str(pd_name), upf_get_value_str(state_name));
```

3. Assertion details: Once we have the handle of a low-power object and its dynamic value it is further passed to a checker module. The checker module can be modeled in system Verilog and compiled together with the design. This module is instantiated in the testbench. The low-power object extracted from information model and its dynamic properties are passed in the interface of this checker instance. We can instantiate as many checker instances as required.

   Consider the example where the requirement is that the combination of state ON for PD_CAMERA and state ON for PD_VIDEO cannot be true at the same time. This can be checked easily using the proposed approach.

   We can define an assertion checker module and create an instance of this module in testbench.

   Assertion Module

```
module assertionPowerState (int state_ON_CAMERA, int state_ON_VIDEO)
reg cov_clk = 0;
always @(state_ON_CAMERA, state_ON_VIDEO)
      cov_clk = 1'b1;

always @(cov_clk)
      cov_clk = 1'b0;

always@(posedge cov_clk)
      assert (state_ON_CAMERA != state_ON_VIDEO) else $error("Camera 
and Video both on at same time");
endmodule
```

<u>Getting the handle & properties of UPF objects</u>

```
// Native HDL objects for power states
upfPowerStateObjT ps_state_ON_camera;
upfPowerStateObjT ps_state_ON_video;

// Handle to hold active state
upfHandleT state_ON_active_camera;
upfHandleT state_ON_active_video;


//integer values of active states, 1 indicates active
integer state_ON_Camera;
integer state_ON_VIDEO;

// continuous access of power states
upf_create_object_mirror ("/tb/chip_top/PD_CAMERA.ON",
"ps_state_ON_camera")
upf_create_object_mirror ("/tb/chip_top/PD_VIDEO.ON",
"ps_state_ON_video")


always @(ps_state_ON, ps_state_OFF, ps_state_SLEEP) begin
      state_ON_active_camera = upf_get_object_properties
(ps_state_ON_camera.handle, UPF_IS_ACTIVE);
      state_ON_CAMERA = upf_get_value_int(state_ON_active_camera);
end
always @(ps_state_ON, ps_state_OFF, ps_state_SLEEP) begin
      state_ON_active_video = upf_get_object_properties
(ps_state_ON_video.handle, UPF_IS_ACTIVE);
      state_ON_VIDEO = upf_get_value_int(state_ON_active_video);
end

Testbench
assertionPowerState checker1(state_ON_CAMERA, state_ON_VIDEO);
```

Using the above approach, any custom low-power checker can be implemented in the design.

VI. Conclusion

The low power designs today are incredibly complex with intricate power architecture. A thorough low power verification is a must for such designs as any power bug left behind can cause huge setback. In this paper we have discussed in-depth the common and custom problems in low-power design and verification. We have also demonstrated how the proposed low-power verification method based on dynamic simulation and coverage can catch these low-power design bugs and help in solving them.

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] "Mohit Jain, Amit Singh, JSS Bharat, Amit Srivastava, Bharti Jain", Overcoming barriers in Power Aware Simulation, DVCon Europe 2014
[4] IEEE Std 1801™-2018 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 29 March 2019
[5] "Awashesh Kumar, Madhur Bhargava", Random Directed Low Power Coverage Methodology: A Smart Approach to Power Aware Verification Closure, DVCon USA 2017