Moving beyond Assertions: An Innovative approach to Low-Power Checking using UPF Tcl Apps

Madhur Bhargava (Madhur_bhargava@mentor.com), Mentor, A Siemens Business

***Abstract* - As the SoCs are getting more and more complex, their effective verification is becoming highly difficult task. The low-power architecture used in today's chips are highly sophisticated and the future trend is only going to go in the same direction. It is now very crucial to catch any power bugs early in the design cycle. Unified Power Format (UPF), the IEEE standard for low-power specification is constantly evolving to address the low-power requirements of the designs. One of the traditional yet effective way to verify the design is to write SV assertions to ensure that right design behavior is met. Verifying the low-power intent of the design using SV assertions is not straightforward because of the fact that there is disconnect between the traditional RTL and low-power objects. Users can not access and manipulate the low-power object in the same way as they do for RTL. To address this issue IEEE 1801-2015 (aka UPF 3.0) introduced the concept of a low-power information model which can greatly simplify the life of verification engineers by providing HDL and Tcl APIs to access and manipulate the low-power information. In this paper we are going to demonstrate with relevant examples and case studies that how we can leverage UPF 3.0 information model TCL query functions (aka Tcl Apps) and tool provided CLI commands to do low-power checking of the design. This is an innovative way of dynamically verifying the low-power intent after the simulation has completed and all the waveforms are available. The paper also explains how users can write their own checker apps for a specific scenario.**

Keywords - Power Management, Low-Power Verification, Low-Power Assertions, Low-Power Dynamic Verification

I. Introduction

The effective verification of low-power designs has been a challenge for many years now. The IEEE 1801 standard for modeling low-power objects and concepts is continuously evolving to address the low-power challenges of today's complex designs. One of the traditional yet effective way to verify the design is to write SV assertions to ensure that right design behavior is met. Verifying the low-power intent of the design using SV assertions is not straightforward because of the fact that there is disconnect between the traditional RTL and low-power objects. Users can not access and manipulate the low-power object in the same way as they do for RTL. Low-power concepts are abstract and the sources which form the low-power information are varied e.g. UPF, HDL and Liberty. Various tool vendors provide tool-generated low-power assertions to dynamically verify the design. Often these tool generated assertions do not suffice the user's need and there is a requirements for additional checks. Users have also adopted varies ad-hoc approaches to write assertions in a checker module. One of these approach is to access the low-power objects using UPF commands and bind these checker modules into the design [2]. However as these checkers are written and compiled along with the design, they have to undergo the whole simulation process. Even a simple task of adding more assertions involves re-iteration of whole low-power simulation flow which is highly time inefficient specially when the simulations are run for larger SOCs.

Low-Power information model was introduced in UPF 3.0 which allows users to access and manipulate the low-power objects. Once the handle of low-power objects is available, a lot of operations can be performed on them. In this paper we are going to demonstrate with the help of relevant examples and case studies that how we can leverage UPF 3.0 information model TCL query functions (aka Tcl Apps) and tool provided CLI commands to do low-power checking of the design. This is an innovative way of dynamically verifying the low-power intent after the simulation has completed and all the waveforms are available. The paper also explains how users can write their own checker TCL procedures for a specific scenario.

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 which 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:

1. 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.

2. 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).

3. 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.

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

5. 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

6. 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

II. UPF 3.0 Low-Power Information Model

UPF 3.0 has come up with the concept of an information model to represent the low-power objects and concepts in a structured and consistent manner. This information model captures the low-power management information. This is the result of application of low-power UPF concepts and commands on the designs. It consists a set of objects and various information-bearing properties defined for those objects. It also defines the relationship between the HDL and UPF. It provides a set of well-defined APIs to query the low-power information in either Tcl or in HDL.

UPF 3.0 information model Tcl APIs can be used to query the static information of a low-power object, e.g. file/line detail of a UPF object or a list of isolation strategies of a power domain and other similar things. To get the dynamic information, we can rely on Tcl APIs provided by the verification tools (simulators) to access the dynamic values of the UPF and RTL objects. Together with the static and dynamic information, innovative applications can be written to help with the checking and debugging of the design.

UPF 3.0 also presents the HDL package functions and native HDL object definition for the UPF object which has some dynamic information, e.g. power domain, power states, etc. Native object definition and usage has been given in the example in the following section. Using these HDL package functions the user can access the static and dynamic information of low-power objects in HDL. This capability can be leveraged to help verification engineers create random verification scenarios.

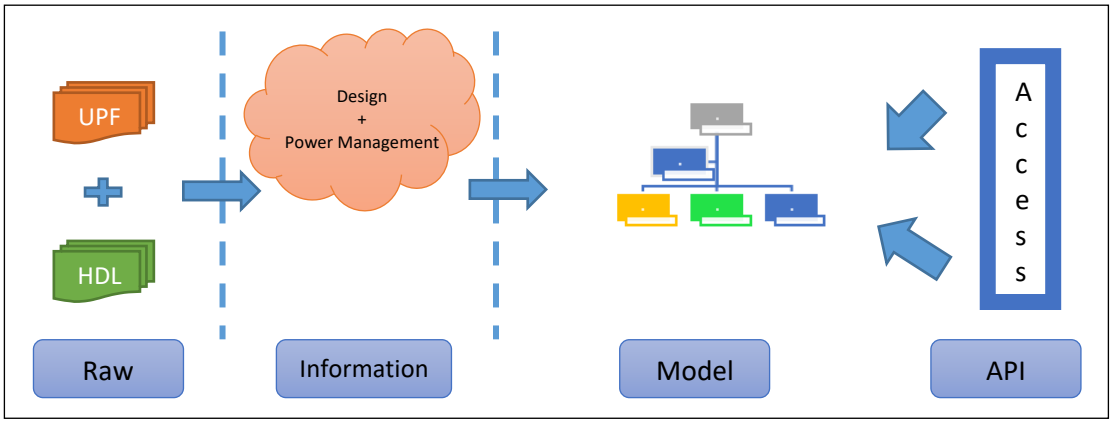

Figure1

There are two main components of the information model.

A. *Objects:* These are the primary holders of information, accessed by handle ID. They represent UPF, HDL and the relationship between them. There are three main classes of objects, namely:
   - UPF Objects: Model objects created by UPF.
   - HDL Objects: Model objects representing the HDL design.
   - Relationship Objects: Objects that model the relationship of UPF and HDL objects, e.g. `upfExtentT, upfCellInfoT`.

B. *Properties:* These are the basic pieces of information, accessed by property ID, such as `UPF_NAME`, `UPF_ISOLATION_STRATEGIES`.

III. Traditional ways of Low-Power Verification

Verification of power management techniques is a challenging task and if not executed properly it can waste a lot of design verification cycles. Low-power verification is generally performed in two steps. First static verification is done to catch all structural errors which include detection of correct placement and connection of power management objects like isolation, level shifter, repeaters and retention. It is followed by dynamic verification where design along with power management architecture is simulated with both functional and power control inputs to detect all control sequences and protocol related errors. This paper focuses on dynamic verification of low-power designs.

Some of the common dynamic verification items in low-power designs are as follows:

- Control Sequence and Protocol checking: In a low-power design, various power management cells like isolation cells, level shifter cells, and retention cells may be inserted into the design which affects the the functionality of the design. Therefore it is of high importance to check that the proper control sequence is followed and these get enabled and active at the proper time. Some of the protocol checks are as follows
  - One of the protocol checks for isolation is to verify that Isolation enable is triggered before the power of the source logic goes down; it remains active throughout the power down period and until sometime after the power goes up.
  - When isolation cells get inserted into the design, the control signal of isolation cell should always remain in a valid state that is either ‘0’ or‘1’. If the control signal goes ‘x’ then the output of isolation cell can go into an invalid state.
  - Similarly if the power domain exhibits retention capability, then another protocol check is to verify that its control signals are triggered at the right time i.e. retention SAVE is triggered before power down and RESTORE is triggered after power up.

- Power Intent checking: Another aspect of low-power verification is related to verifying UPF intent against the implemented power aware design. For example user wants to ensure that during the active isolation period, the signal on which isolation is applied is being clamped to the correct clamp value as specified in UPF file.

A. Assertion based low-power verification

One of the powerful ways to verify the design is to write SV assertions to ensure that right design behavior is met. This has been traditionally used in non-low-power simulations for quite a long time. With the advancements in low-power verifications, SV assertions provide a good way to achieve the dynamic verification of low-power designs as described in [2]. SVAs are used to validate power control logic sequence and also ensure that specific requirements are met before and after power mode transitions. These assertions can monitor and check transitions in the power control signals to verify legal and correct low-power behavior. Verification of the low-power designs using system verilog assertions is met in following two ways:

1) Tool provided dynamic low-power verification

One of the most common ways to do low-power verification of the designs is to rely on simulation tool provided dynamic verification capabilities. Some of the EDA vendors provide automated, tool-generated assertions to check common protocol errors. These are quite powerful as they are built-in and generated by the simulation tools so these are often fast and consume less simulation cycles as compared to traditional SVAs. Following are some of the challenges with this approach of low-power verification.

- Doesn't suffice all needs: Quite often these are not complete, the user come up with new requirements and checks which have to be added in the tool thus leading to delays.
- Slow Performance: Another challenge is that when compared to a design run without enabling the tool provided assertions, these result in slow performance and it cannot be turned on in the default flow.

2) Custom Assertions

To model the custom needs, users have to rely on their own custom low-power assertions to verify the power intent of the design and also make sure that all test scenarios are covered. Modeling these low-power assertions in HDL in conjunction with their power intent is a very complex task, and is achieved by using the bind_checker command and methodology as defined in [2]. Following are some of the challenges with this approach of low-power verification.

- Complexity: Custom assertion flow is little complex to use as users have to rely on mix of UPF commands and assertions models written in HDL which may not be straightforward.
- Slow Performance: Another major challenge in custom assertion flow is that it leads to slow performance of the simulation runs making it difficult for users to run the assertion flow in their regressions.
- Multiple Flows: These assertions are written in a separate module and instantiated inside testbench. As a result, users end up having two flavors of testbench, one for regression flow and another for low-power checking flow.

IV. Tcl Based Low-Power Checking Methodology

As seen in the above sections, two major challenges associated with low-power verification using system verilog assertions is to allow users write their own custom checkers and faster performance to lower the simulation cycles. To address these challenges the idea of post simulation low-power checking of the design based on UPF Tcl Apps came up. This low-power checking methodology involves running the simulations without any SV assertions and checking of design is done post-sim mode where all the waveform data along with low-power information is available. Tcl based checker applications are Tcl procedures that users can write for special requirements of low-power checking of the design.

Building blocks of low-power checker applications include:

- UPF 3.0 Information Model APIs: Information model provides four basic APIs which can be used to access any UPF object, gets its handle and associated information.
- Tcl APIs (Waveform CLIs) provided by verification tools (simulators) to access the dynamic data

The methodology involves running the simulation in normal regression model. During the post simulation mode, low-power design objects are accessed along with their value information to write specific checker scenarios. As it methodology has access to full waveform information, it enables users to do structural checks and protocol related checks.

A. Steps required for methodology

Once the simulations are completed, the waveform data is loaded simulator's interface and TCL CLIs/APIs are used to access this wavedata.

- Write the testbench without any low-power assertions
- Run the low-power simulations in regression mode maximizing the simulator performance
- Allow generation of low-power information model database so that we can access the low-power objects later during the verification flow
- Wavedata is logged for both RTL and low-power objects defined in the UPF file

- Launch the simulator visualization / batch interface to load the low-power information model database along with the design data and the wavedata
- Low Power Checker Applications
    - Describe the intent of the checker application, that is the specific scenario which needs to be verified in the design run
    - Write a low-power checker application which can then be run in the simulator's interface
    - Low-Power checker app relies on Tcl APIs to access low-power objects and simulator CLIs to access the wavedata
- Run the low-power checker app

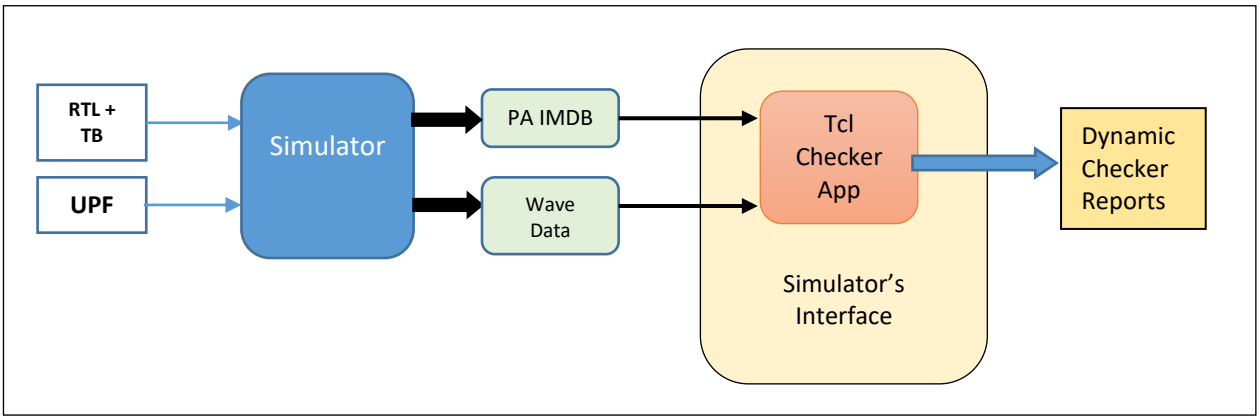

Figure2

B. General steps for writing any checker application

- Define the intent of the checker application: Intent involves writing the description of the control sequence or protocol
- Tcl Application involves iterating over the design or power domains to access and get the handle of relevant low-power objects
- Access the waveform information of these objects
- Write the condition/expression to ensure the checker intent
- Flag a message in case of any violation

Using the Tcl APIs and waveform CLI commands, a number of novel low-power checks can be performed as described in the case studies.

V. Case Studies

Here we look at some of the checker examples that can be built using the above mentioned methodology. By using the `alias` tcl command, some alias names for long tcl procs are created and thus referred by their alias names in the example code.

```
alias query upf_query_object_properties
```

A. Example1

In a low-power design, if the driving logic domain goes into an OFF state while the receiving logic is still in normal mode of operation, then there is a need of an isolation cell between the driving and receiving logic. During this period, isolation cell is enabled and output of the isolation cell is clamped to known a value. If somehow supply of an isolation cell is turned off during this period, then it can result in an unexpected corrupted value on the isolation output and design will be functionally incorrect. In this example the goal is to check and find any isolation strategy in the design whose supply is off at the time when isolation strategy got enabled.

Step 1: Get list of all power domains in the design and iterate over them

```
foreach pd [query_power_domain *] {  …   }
```

Step 2: Iterate over all the isolation strategies of a power domain

```
foreach iso [query $pd –property upf_isolation_strategies] {
   array set iso_details [join [query $iso –verbose]]
}
```

Step 3: Query isolation strategy to get the handles of its isolation enable, signal sense and power/ground supplies

```
set isosig [query $iso_details(upf_isolation_controls) –verbose]
set iso_power $iso_details(upf_isolation_power_net)
set iso_ground $iso_details(upf_isolation_ground_net)
```

Step 4: CLI commands to access UPF objects wavedata and perform checks

```
set vcHandle [vpi_handle_vc $isosig $waveFileHandle]
set waveValueIter [vpi_iterate_vc $vcHandle $waveMin $waveMax "'b*0*"
"'b*1*"]
while {[set valueHandle [vpi_scan $waveValueIter]]} {
 set timeNs [vpi_get_str vpiTime $valueHandle]
 set pwr_val [examine $iso_pwr -time $timeNs]
 if {$pwr_val == "0"} {
  puts "PA_Check_Error: $timeNs ns, Isolation Power is off when Isolation
Strategy '$iso' got enabled"
 }
…
```

These steps are written in a tcl procedure (aka PA App) which can be called from the simulator's interface. Following is the final tcl proc.

```
proc pa_checker_iso_power_at_enable {args} {
  foreach pd [query_power_domain *] {
    foreach iso [query $pd -property upf_isolation_strategies] {
       array set iso_detail [join [query $iso -verbose]]
       array set iso_ctrls  [join [query $iso_detail(upf_isolation_controls) -verbose]]
        set isolation_signal $iso_ctrls(upf_control_signal)
        array set oData [join [query $isolation_signal -verbose]]
        set isolation_signal [string trimleft [regsub -all "/" $isolation_signal "."] "."]
        set isolation_sense  $iso_ctrls(upf_signal_sensitivity)
        set iso_pwr $iso_detail(upf_isolation_power_net)
        set iso_gnd $iso_detail(upf_isolation_ground_net)
        set lvar1 [vpi_handle_by_name $isolation_signal 0]
        set vcHandle [vpi_handle_vc $lvar1 $waveFileHandle]
        if {$isolation_sense == "high"} {
          set waveValueIter [vpi_iterate_vc $vcHandle $waveMin $waveMax "'b*0*" "'b*1*"]
        } else {
          set waveValueIter [vpi_iterate_vc $vcHandle $waveMin $waveMax "'b*1*" "'b*0*"]
        }
        while {[set valueHandle [vpi_scan $waveValueIter]]} {
          set timeNs [vpi_get_str vpiTime $valueHandle]
          if {$iso_pwr != ""} {
             set pwr_val [examine $iso_pwr -time $timeNs]
             if {$pwr_val == "0"} {
               puts "PA_Check_Error: $timeNs ns, Isolation Power is off when Isolation Strategy
'$iso' got enabled"
             }
          }
          if {$iso_gnd != ""} {
            set gnd_val [examine $iso_gnd -time $timeNs]
            if {$gnd_val == "0"} {
              puts "PA_Check_Error: $timeNs ns, Isolation Ground is off when Isolation Strategy
'$iso' got enabled"
            }
          }
       }
```

```
    }
}
```

B. Example2

When any PA cell gets inserted into the design, it is expected that the control signal of that PA cell will be in a valid state of either '0' or '1'. For example, if the isolation signal has value "0" then isolation cell is disabled and input value is driven to isolation output and when the isolation signal has value '1' then isolation cell is enabled and output of isolation cell is clamped. However if the isolation signal has a value 'x', then output of cell is unknown. In this example the goal is to check and find any isolation strategy in the design whose isolation control gets corrupted.

Tcl Proc

```
proc pa_checker_iso_ctrl_crpt {args} {
#get all domains
    foreach pd [query_power_domain *] {
#get all isolation strategy
      foreach iso [query $pd -property upf_isolation_strategies] {
        array set iso_detail [join [query $iso -verbose]]
        array set iso_ctrls  [join [query $iso_detail(upf_isolation_controls)
-verbose]]
#get handle of isolation signal
        set isolation_signal $iso_ctrls(upf_control_signal)
        set lvar1 [vpi_handle_by_name $isolation_signal 0]
        set vcHandle [vpi_handle_vc $lvar1 $waveFileHandle]
        set waveValueIter [vpi_iterate_vc $vcHandle $waveMin $waveMax "*"
"*"]
#iterate over value change of isolation signal
        while {[set valueHandle [vpi_scan $waveValueIter]]} {
          set timeNs [vpi_get_str vpiTime $valueHandle]
          set val [examine $isolation_signal -time $timeNs]
#check if isolation signal has value 'x'
          if {$val == "1'bx"} {
      puts "PA_Check_Error: $timeNs ns, Isolation Control of Isolation
Strategy '$iso' got corrupted"
          }
        }
      }
   }
}
```

C. Example3

For any design it is important to check domain activity and may report the power domains that are switched on or off. It might be useful to report any power domains which remain always on or always off during the entire simulation run.

Tcl Proc

```
proc pa_checker_domain_activity {args} {
#iterate over all power domains
    foreach pd [query_power_domain *] {
        set name [query $pd -property upf_name]
        set timeChange [getWaveValueChange $pd "*" "*"]
#getting the activity count for power domain
        set activityCount [expr [llength $timeChange] -1]
        if {$timeChange == 0} {
            set activityCount 0
        }
#can check and flag message if activity count is zero
        echo "+Power Domain '$pd' (Activity Count - $activityCount)"
}
```

D. Example4

One of the extension of above procedure could be to report any voltage modulation on the power domains as it is used in the DVFS techniques.

```
proc pa_checker_domain_activity2 {args} {
#iterate over all power domains
    foreach pd [query_power_domain *] {
        set lvar1 [vpi_handle_by_name $pd 0]
#get waveform handle of power domain
        set vcHandle [vpi_handle_vc $lvar1 $waveFileHandle]
        set waveValueIter [vpi_iterate_vc $vcHandle $waveMin $waveMax "*"
"*"]
#iterate over state/voltage change of power domain
        while {[set valueHandle [vpi_scan $waveValueIter]]} {
          set timeNs [vpi_get_str vpiTime $valueHandle]
          set val [examine $pd -time $timeNs]
#report state/voltage change of power domain (primary/ground supply)
            puts "PA_Check_Report: $timeNs ns, Power Domain changed
state/voltage to $val"
          }
        }
    }
}
```

VI. Benefits over conventional approaches

The proposed methodology in this paper using UPF 3.0 information model provides a number of benefits and can be used to achieve an early low-power verification closure. Following are some of the benefits of using the proposed methodology over the conventional approaches of assertion based low-power verification.

- Flexible and easy to use: The defined approach is flexible and easy to use as it keeps the checker module accessing UPF objects and original RTL separate. Conventionally users write two testbench, one for the regression mode and other having custom low-power assertion methodology code. As the defined approach is done at post-sim time, the simulation can be performed using the testbench used in regression mode avoiding the need for any additional testbench.
- Better performance: Whether it is enabling the tool based assertions or relying on custom assertions, it has performance overhead compared to normal regression mode. Since this approach is post-sim based, the regression mode is sufficient to power low-power checking leading to better performance.
- Inter-operability: It is consistent across tool vendors as it is based on UPF 3.0 standard and all the simulators provide CLI interface to access waveform information.
- Easy to scale up: It is based on Tcl code and make use of information model Tcl APIs. Whenever there is a need for a new check, it is easy to code in Tcl and run on the design. Overall development and verification time greatly reduced.

The proposed methodology is not completely full proof and has some drawbacks as well.

- Not every check can be coded in TCL: Though low-power information model has information about all UPF objects and allows to access and manipulate them, it lacks the information about design connectivity. Some of the checks involving design connectivity like source to sink path may not be straight forward to code in Tcl due to lack of connectivity information in UPF 3.0 information model. If this information comes into database, then such checks can be performed as well.
- Simulation keeps running: SVAs have a clear advantage that if any fatal error or assertion gets triggered during the simulation run, the run would stop at that point itself and won't continue further. This prevents unnecessary simulation cycles. However since these are tcl based applications these do not monitor the activity during the run and can just check at a particular time, so they cannot be run in live sim mode.

As power intent of mixed-signal designs is out of scope of UPF standard itself, this has not be covered in the paper as well.

VII. Conclusion

The low power designs today are incredibly complex with intricate power architecture. A thorough low-power verification is must for such designs as any power bug left can cause huge setback. In this paper we have discussed the challenges with the current low-power checking of the design and how the proposed method can be address this better with the help of UPF 3.0. We also presented with examples of low-power checkers (Tcl code) to address the problems in low-power verification. These case studies also demonstrate how users can write their own low-power checkers to develop a more consistent, robust and scalable low-power verification platform. In the end we discussed the benefits of using the proposed approach over conventional approaches.

VIII. References

[1] IEEE Std 1801™-2015 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 05 Dec 2015.
[2] “Madhur Bhargava, Durgesh Prasad”, Low-Power Verification Methodology using UPF Query functions and Bind checkers, DVCon Europe 2014
[3] “Amit Srivastava, Awashesh Kumar”, PA-APIs: Looking beyond power intent specification formats, DVCon USA 2015