2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Using UPF Information Model APIs to write re-usable Low Power Testbenches and customized coverage models for Low Power

Shreedhar Ramachandra
Synopsys Inc., Mountain View
Himanshu Bhatt
Synopsys Inc., Mountain View

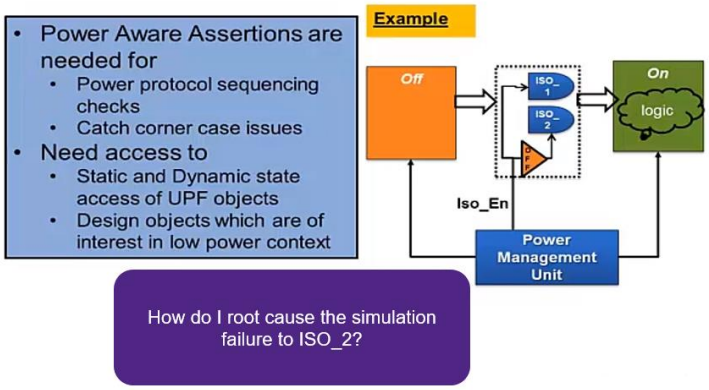
***Abstract*- Low Power Testbenches with UPF today are mostly designed to have a one-way communication with the UPF objects with no way to monitor their state. The UPF Information model APIs provide a consistent way to access UPF information at compile time (using TCL queries) and runtime (using System Verilog APIs). This paper explores the applications of the UPF Information model APIs namely writing reusable Low Power testbenches with the ability to monitor and control the UPF objects and build customized coverage models for Low Power. The possible impact on simulation performance is also examined.**

I. INTRODUCTION

Low Power Testbenches today have no visibility of the UPF objects and their states during a Low Power simulation. This has been one of the factors limiting the users from writing re-usable Low Power testbenches that can monitor the UPF objects and react to the state changes of UPF objects. To meet this requirement for the user to query the UPF datamodel and the states of the UPF objects during a Low Power simulation, the UPF LRM has defined the UPF Information Model and added the mechanism to query the UPF datamodel at compile time (using TCL queries) and at runtime using (System Verilog APIs). UPF Information model is the definition of how the UPF information is organized in the form of a UPF datamodel that can be queried using TCL and System Verilog. The TCL query mechanism lets the user query the UPF Information Model. The System Verilog APIs to can be used to query the UPF datamodel and monitor and control the UPF objects during simulation. This paper explores two possible applications of the UPF Information Model APIs that will enable the users to write re-usable Low Power testbenches that can monitor and control the UPF objects. The third application of these APIs that is described is to create customized coverage models for Low Power coverage of UPF objects. The last section describes the possible impact on simulation performance with the use of compile time and runtime access mechanisms. The UPF Information model that is a part of the UPF standard is still undergoing enhancements to cater to more requirements that the users have in terms of querying the UPF Information. The scope of applications that can be built on these APIs is much more than what is described here.

II. NEED FOR LOW POWER ASSERTIONS

Assertions (both in design and test bench) are an integral part of dynamic verification methodology. With low power their significance increases multifold because of the complexity of the current and NextGen low power SoCs. They enable a reduced TAT in the debug and overall verification closure effort.

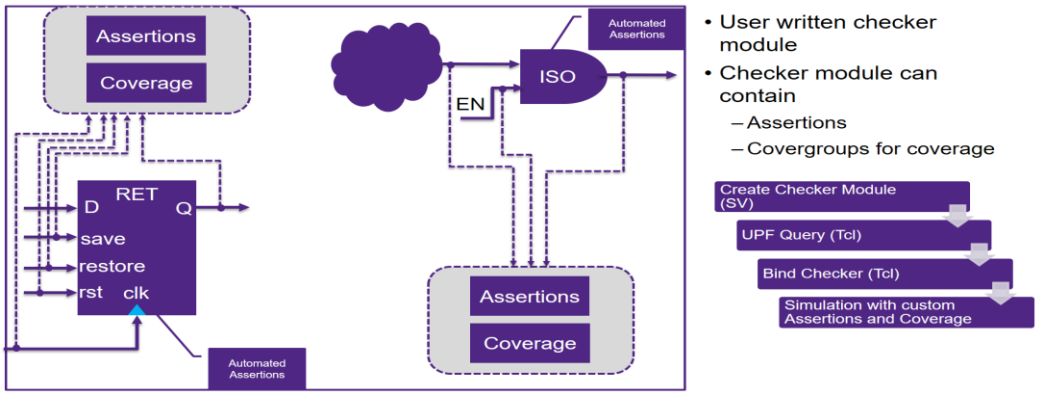


III. APPLICATIONS USING UPF INFORMATION MODEL APIS

A. Bind Checker using UPF TCL Queries

The UPF TCL queries are now designed to traverse the entire UPF Information Model and access every object that has been created and its relationships with the other UPF objects. This mechanism along with bind_checker UPF command can be used to bind custom checkers containing System Verilog assertions and properties. These checkers bind to UPF objects during the simulation to cover the properties or trigger the assertions.

Power Aware Verification Environment (PAVE) is an infrastructure that enables accessing the UPF objects, monitor low power events, and write power-aware assertions. It uses the powerful UPF query commands to query the power intent and UPF bind_checker command to bind the checker modules to the UPF objects like power domains, power switch, isolation strategy, and retention strategy. User can use the PAVE infrastructure to write custom assertions for scenarios like “clock parked high” when “reset is low” etc.



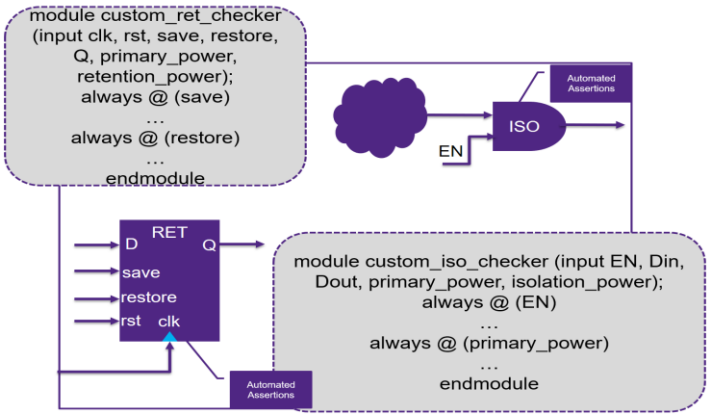


Figure 1: PAVE infrastructure

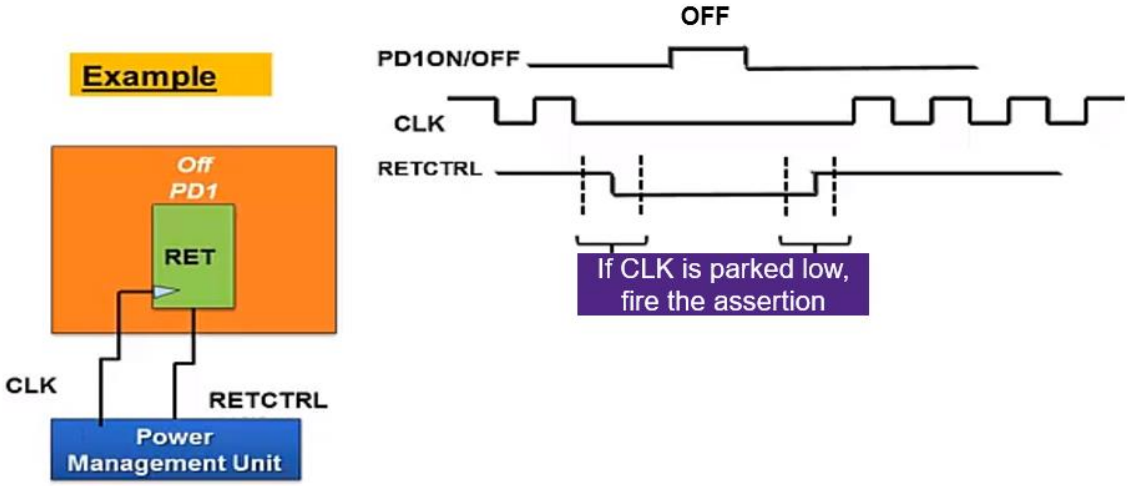


Figure 2: Custom assertion scenario to check that the clock should not be parked low during save and restore operations

B. Re-usable and Reactive testbenches using System Verilog APIs

PAVE methodology described in the previous section gives user the flexibility to write custom assertions based on the different scenarios which are related to specific design scenarios related to MV cells like isolation, retention etc. Low Power testbench enables the user to be able to control the power aware simulation by being able to "probe" the UPF supply network e.g. a particular PST state and based on that either change the stimulus or write an assertion to verify a scenario. IEEE UPF LRM lists down several such APIs which can be used to achieve this. The power aware simulator must support these APIs to enable the user to use this functionality.

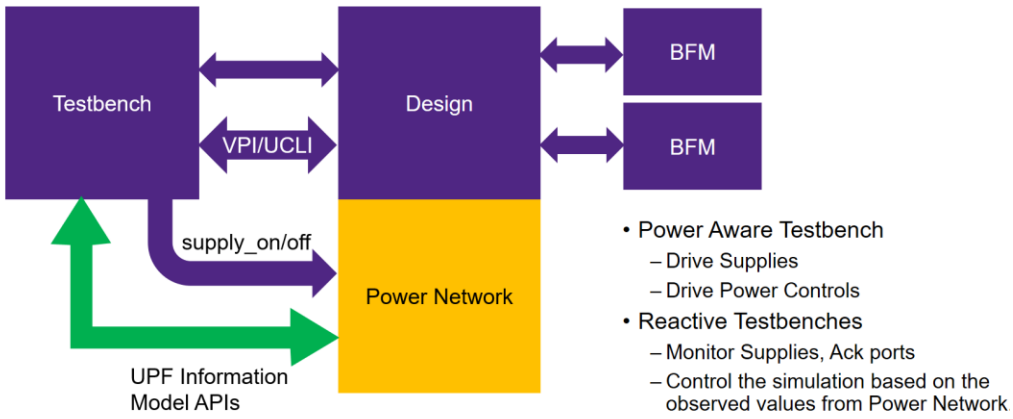


Figure 3: Low Power testbench (two-way information exchange)

The functionality of a Low Power testbench is to drive the top level supplies and modify the Low Power control signals (isolation control, retention control and power switch control signals). Currently users write targeted testbenches for Low Power that do these tasks but have no visibility into the actual simulation as to whether these actions have been successful or not. Moreover, the testbench has no way to monitor the states of the UPF objects during simulation or react to those state changes. The System Verilog APIs for the UPF Information model provides a mechanism for the user to traverse the entire UPF hierarchy in the System Verilog testbench and monitor the UPF objects. The user testbench can take actions based on the state changes of these UPF objects.

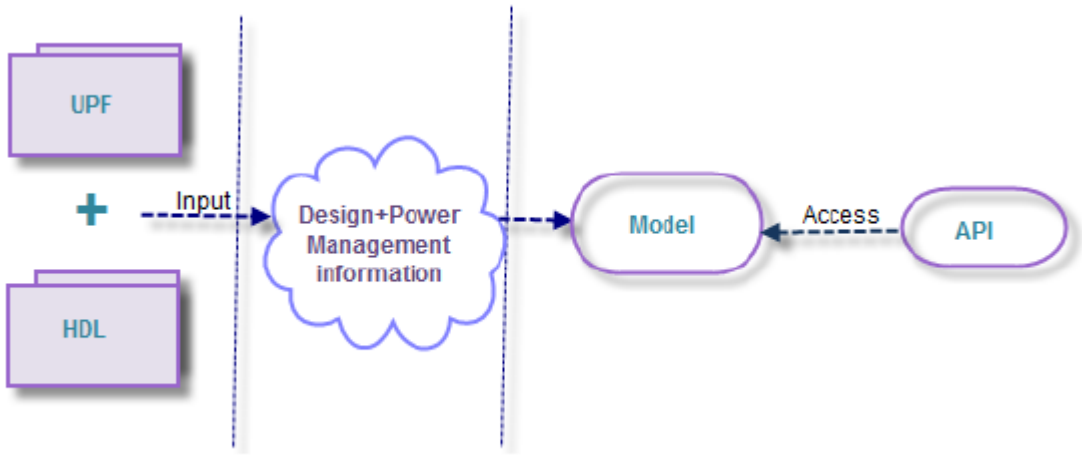


Figure 4: UPF Information Model flow

The UPF information model captures the power management information which is the result of application of UPF commands on the user design. It consists of a set of objects containing information and various relationships between them.The UPF information model consists of a collection of objects. The information is present on these objects in the form of properties, which can be accessed via APIs. Each object is denoted by a unique identifier (ID) called a UPF handle which is used by the APIs to access information present on it. The objects and properties in the information model can be accessed by a HDL object of *upfHandleT* type. The class IDs are unique strings that

represent class names in the information model. The class names are represented in capitalized words with a upf prefix and T suffix.
Example: *upfPowerDomainT, upfHdlScopeT*
The property IDs are unique strings that represent the properties present on the object. The property names are denoted by all lowercase words separated by an "_" (underscore character) and a upf prefix.
Examples: *UPF_NAME, UPF_PARENT*
The HDL interface to the information model allows user to create abstract testbenches to manipulate UPF objects directly in simulation.

C. Customized coverage models for Low Power coverage.

Low Power Verification today rely on simulators to do some automated coverage of UPF objects. As explained in the previous section the System Verilog APIs can be used to access the traverse the UPF information model and monitor the states of UPF objects. The user can write covergroups and coverpoints that monitor the states of the UPF objects to do their own coverage which can be customized based on the user requirement.

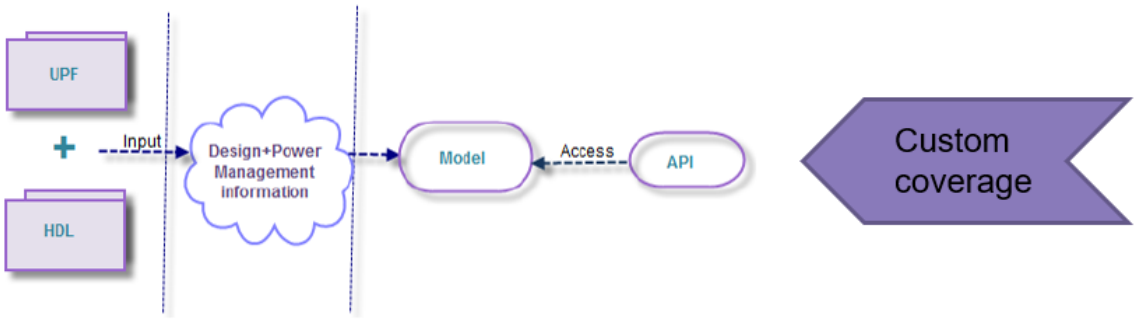


Figure 5: Custom coverage

III. IMPACT ON SIMULATOR PERFORMANCE

The usage of either the TCL queries at compile time or the System Verilog APIs at runtime has an impact on the simulator performance. There is lot of functionality that is common between the two access mechanisms. The choice of whether to use the compile time queries or the runtime System Verilog APIs should be made based on the following considerations.

a) The compile time TCL queries should be preferred over the runtime APIs since the user would prefer a degradation in compile time when there is a tradeoff between the compile time and runtime.
b) There are some applications that can be built only using the runtime APIs, for example monitoring states of UPF objects during simulation. However, the amount of performance impact is proportional to the number of UPF objects whose states are being monitored at runtime.

REFERENCES

[1] 1801-2015 - IEEE Standard for Design and Verification of Low-Power, Energy- Aware Electronic Systems. Section 10 and 11.