An Integrated Framework for Power Aware Verification

Harsh Chilwal
Synopsys India Pvt Ltd
Bangalore, India
Email: harsh@synopsys.com

Manish Jain
Synopsys India Pvt Ltd
Bangalore, India
Email: mjain@synopsys.com

Bhaskar Pal
Synopsys India Pvt Ltd
Bangalore, India
Email: bpal@synopsys.com

***Abstract*—Recent advances in UPF (Unified Power Format) have enabled ways to query the power objects from the UPF (power architecture) data base, and ways to bind checkers without affecting the actual design. In this paper we attempt to leverage these strengths of UPF to ease the specification of power intent (properties/assertions) and instantiation of these in the design for verification. We show that the proposed framework enables the verification engineer to specify its low power intent conveniently in a unified manner and the same intent can be used across different levels of abstractions seamlessly. Though we show the flow using UPF, the methodology is generic and can be implemented/supported in any other power formats.**

I. Introduction

Typically in a low power design flow, the design is partitioned into a set of power domains. The power network of the design is then modeled using power elements like isolation, retention, power switch, level shifter cells, supply nets, port etc. In addition to these, there exists power management units (PMUs) to control this power architecture by issuing appropriate control sequences to the different elements of the power architecture. Application of control sequences forces the design to change its power states (examples of states include - on, off, stand-by, etc).

In general, both the power architecture and the PMU functionality (in terms of state machines) can be specified using languages like UPF [3]. This specification can exist in the individual power domains as well as at a global (top level) scope. We will call these as - Local Power Management Units (LPMU) and Global Power Management Units (GPMU). Once a set of UPFs for a design are read by a tool, an internal power architecture data base is created which resembles intent of these set of UPFs.

In addition to its capability to specify the power architecture, UPF also offers features to help verification of this functionality. For example, recent advances in UPF standard have enabled people to query the power architecture (UPF) data base. Moreover commands like bind_checker enables attaching checkers/properties without changing the functionality of the design.

Once this power intent specification is over, the power architecture verification for a given low power design starts and it involves the following steps.

1) First, we need to verify that the power management architecture is structurally correct.
2) Second, we need to verify that the design behave correctly when power management control signals are given in the correct sequence.
3) Third, we need to verify that the power control logic will always generate power control signals in the correct sequence.

The first step is usually carried out by static checking tools like MVRC [5], which usually do a correctness and consistency check of the UPFs (written for a design). The other steps are generally executed during simulation [1], [2], [4] starting at an early stages (e.g RTL) of the design flow. Low power architecture verification using simulation is tricky and its success depends on two key components.

1) In the early stages, the power architecture is specified only in the UPF, it does not exist in the design. Therefore, the simulation tools use the UPF to internally synthesize the UPFs in the given design. On the otherhand, the PMUs can be modeled using testbenches or sometimes these can be part of the design as well.
2) The power intent (for verification) is typically modeled using properties/assertions. Modeling this can be very difficult because it includes properties than span a single power domain (in the context of LPMU) as well as multiple power domains (in the context of GPMUs).

In this paper, we provide a methodology which leverages features of UPF to offer a programmable interface to the designers/verification engineers which can be used to verify complicated power assertions/checkers very easily. The properties can be any SystemVerilog Assertion (SVA) [7] and it can be either domain specific (for the LPMUs) or inter-domain type (for the GPMU). Specifically the methodology proposes the following steps:

1) Express the power intent by a set of parameterized SVAs.
2) Once the properties are modeled, the context of attaching these SVAs is modeled by the proposed programmable interface. This interface is designed over the *query_upf* commands.
3) Once the specific context is modeled by this programmable interface, the attachment of the checkers to the power objects is done by UPF bind_checker command.

We have observed that with the proposed flow, the verification process can be hugely automated and the closure can be achieved in less time.

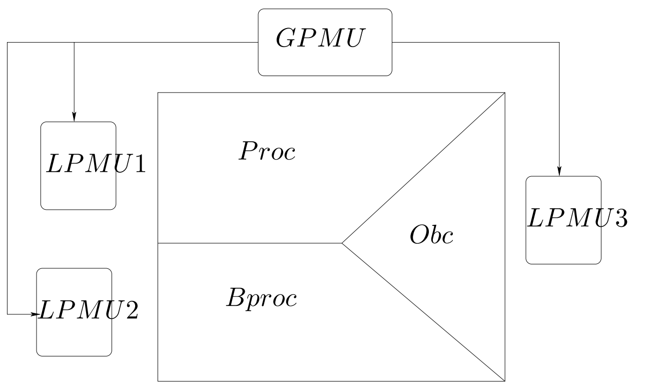

Fig. 1. Example design

II. A Motivating Example

We illustrate the above mentioned steps using an example design in Fig 1. This example is partially motivated by [4]. The design consists of three power domains, namely, a processor (Proc), a backup processor (Bproc) and a On-chip Bus Controller (Obc). The different power modes of these domains are given as follows:

1) The Proc has three power modes - *on*, *off* and *standby*.
2) The Bproc works in three power modes - *on*, *lowpower*, *off*. Lets also assume that there is a design signal *data_rate* that controls the voltage supply of Bproc. The voltage regulator of Bproc switches the supply from *VDDML* (low voltage supply) to *VDDM* (high voltage supply) whenever *data_rate* goes above a predefined threshold (*TH*).
3) The Obc works in two power modes - *on*, *off*.

Both the Proc and Bproc uses the on-chip bus for their data transfer. Proc is the main processor that is active most of the time. When Proc is on *standby* or *off*, the Bproc stars working either using full power (*on* state) or in restricted mode (*lowpower* mode). The Obc can never go into *off* when one of these processors is working.

Each these three power domains has their own LPMUs. Also there is a GPMU which controls interactions between these LPMUs. Note that the states of PMUs are the power states of the design. Therefore transitions between these states may not be immediate, it might take several design clock cycles for transition to a different state. For example, a transition from *on* to *off* state of Proc requires - (a) retention of the current state of Proc, and (b) enabling isolation cells of Proc by activating isolation control signals. These are the called the implicit states in the design.

In Fig 2, we show the power state transition diagram of Proc. We model these implicit states using state *impl1* and *impl2*.

Now let us try to specify some of the power intents of this power architecture that we need to verify.

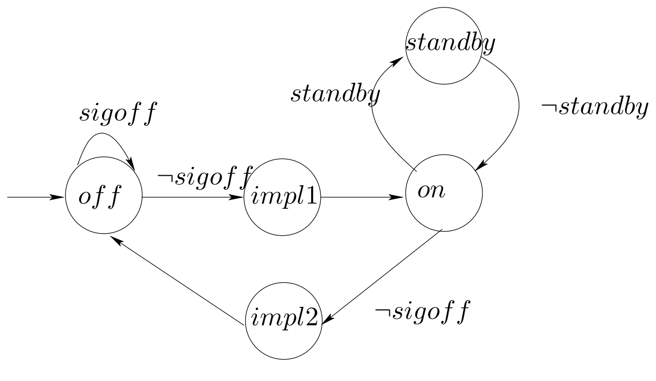

Fig. 2. LPMU of Proc

- Global Properties: The global properties are applicable to all the LPMUs. One such property is *to verify the correct isolation of all the power domains*.
- Local Properties: Properties that are applicable to a specific LPMU. The example properties can be:
    1) Verify that whenever the Proc is in *standby* state, the retention control signals (save/restore) can't be active till Proc gets back to its *on* state. Note that this property is specific to Proc.
    2) A property contains design signals as well. For example, one of the properties of Bproc can be - whenever design signal *data_rate* crosses threshold *TH*, the voltage supply of power domain Bproc should be the high voltage supply *VDDM*.
- Inter-domain Properties: These are to verify the interactions between the LPMUs. Example properties can be -
    1) When Proc switches from its *on* state, Bproc switches to one of its active state.
    2) Both Proc and Bproc are never in the *on* state together.

The above properties are usually high level descriptions using design signals and UPF signals. The designer/verification engineers takes this descriptions and then translate these into some languages like SVA. However, these checkers (in SVA) need to be instantiated into the design properly such that the formal arguments of the properties are replaced by actual design/power signals for verification using simulation/formal engines.

This task of instantiation in very difficult because:

1) Explicit instantiations of properties consisting of power control signals is not trivial.
2) Properties consists of power/design signals across the domains. Moreover properties can be global as well as local. Therefore manual binding of the signals in the property can be erroneous and time consuming.

To address the above mentioned difficulties, we propose a programmable interface that helps to instantiate the properties in terms of the power/design signals that may span signals across the whole power architecture/design in a convenient manner.

III. Programmable interface using UPF queries and bind checkers

The most intuitive way to associate power signals with a checker is through UPF as it acts as the power architecture database manager where we can get all the power related information. However, to access these, as users, all we need is a powerful query interface. Thankfully the current enhancements to UPF provide a powerful set of commands (called **query_upf** commands) for searching design elements related to power architecture, supply nets, supply ports, policies (isolation/retention/power switch) etc. The **query_upf** command works on the logical hierarchy from a domain-centric (in the context of LPMUs) or hierarchy-centric (in the context of GPMU) approach. A domain-centric approach restricts the search to design elements, net, or ports that are logically within the extent of the specified domain_name. A hierarchy-centric approach searches in the scope only, or in and below the scope when -transitive is specified [3].

The current standard of UPF also provide a command called *bind_checker* that helps to instantiate checker modules into a design without modifying the design code or introducing functional changes. The mechanism for binding the checkers to design elements relies on the SystemVerilog bind directive [7]. The bind directive causes one module to be instantiated within another, without having to explicitly alter the code of either. Signals in the target instance are bound by position to inputs in the bound checker module through the port list. Thus, the bound module has access to any and all signals in the scope of the target instance, by simply adding them to the port list, which facilitates sampling of arbitrary design signals. One small syntax example is:

```
bind_checker chk_p_clks
-module assert_partial_clk
-bind_to ahd
-ports {{port1 clknet2} {port3 net4}}
```

However we need to remember that in the early stages of the design, UPF (power) signals are not there explicitly in the design. Therefore some mechanism must be there to specify these signals in terms of UPF names and then internally associate the original design signals with these.

The proposed programmable methodology uses *bind_checker* and *query_upf* commands to automate instantiation of any SVA checker consisting of any design/power signals. The interface is Tcl [8] based and can be embedded in UPF itself. Therefore there is no need to learn a new language.

Now we show how to use this interface for the above mentioned properties. Take out the following property:

A property contains design signals as well. For example, one of the properties of Bproc can be - whenever signal data_rate *crosses threshold* TH, *the voltage supply of power domain Bproc should be the high voltage supply* VDDM.

The above mentioned requirement can be expressed by the following SVA. Here, power_state_simstate is a predefined type in UPF.

```
module checker(d,s,sref,clk);
parameter TH = 1.0; input real data_rate;
input supply_net_type s, sref;
assert property
    (@(clk) (d>TH)|->(s==sref));
endmodule
```

Next this checker is attached to the power-controller using the enhanced bind_checker and query UPF

```
foreach domain {query_power_domain Bproc
                -no_elements -detailed} {
bind_checker data_rate_check \
-module checker -bind_to $domain \
-ports {
        {d data_rate}
        {s $domain.primary.power} \
        {sref $domain/VDDM}
       }
-parameters {TH 3.2}
}
```

Note that we first query about the specific domain and then uses the domain handle to attach the actual power signals to the formal port names.

Now lets take the global property: *Verify the correct isolation of all the power domains.* This property can be expressed by the following SVA.

```
module iso_en(sim_state,en,pw,gnd);
input upf_simstate sim_state;
input en,
input supply_net_type pw, gnd;
import UPF::*;
bit isolation_enable;
assign isolation_enable =
 (((pw.state == FULL_ON)
    && (gnd.state == FULL_ON))
       && en) ? 1’b1 : 1’b0;
always @(sim_state) begin
if (sim_state == CORRUPT)
    assert(isolation_enable == 1’b1);
endmodule
```

This checker can be attached using the enhanced bind_checker syntax and query UPF as follows:

```
foreach domain {query_power_domain *} {
foreach isolation
  {query_isolation * -domain $domain} {
bind_checker $(domain)_$(isolation)_isen \
 -module isolation_en \
 -ports {
  sim_state {${domain}.simstate} \
  {en ${isolation}.isolation_signal} \
  {pw ${isolation}.isolation_power} \
  {gnd ${isolation}.isolation_ground}
 }}}
```

Note that this method of instantiating the checkers enables automated way to attach the checker *in all the domains* in only one step. These steps are described in Fig 3.

In this section, we have used only properties that are checkers. However, other kinds of properties (like coverage properties) can also be attached in this manner. Note that most of the coverage monitors span multiple power domains. Therefore the above programmable interface is an ideal candidate for these coverage properties.

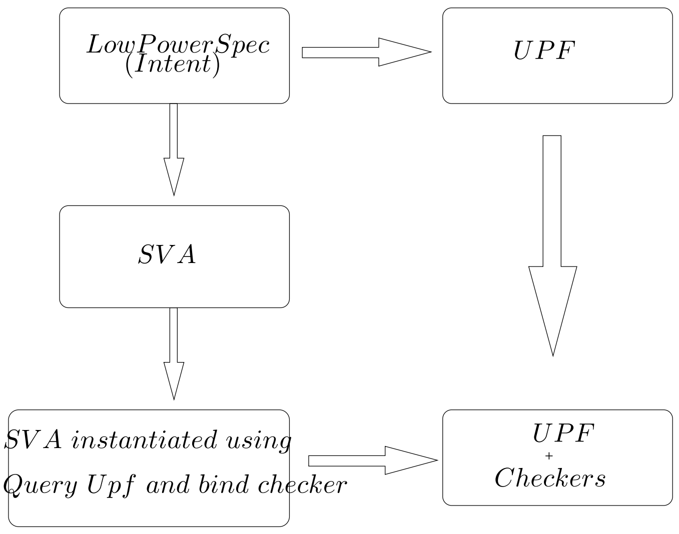

Fig. 3. Proposed flow for translating power intent into power assertions

IV. Tool Flow and Results

Once the above task is done, the simulation tool (like MVSIM [6]), can take the design, UPF and these modified checkers as inputs. Next it internally instruments the design and instantiates the checkers appropriately with the design and/or instrumented UPF signals. The tool flow is shown in fig 4. During simulation it runs the checkers and outputs appropriate messages. The tool also has capability to control the assertions such as enable/disable/coverage etc.

We have seen that the proposed methodology has minimal overhead on the low power verification performance. This overhead comes for automatic binding of the checkers. We are sharing the overhead on only some of the industrial designs(See I). The overhead is more or less same on some large scale designs as well. The 1st col represents the name of the design. The 2nd col shows the number of assertion messages. The 3rd and 4th col show the size of UPF and RTL respectively. The last col shows the percentage runtime overhead.

V. Conclusions

In this paper we have presented a programmable interface using Tcl that attempts to automate the steps to bind low power assertions in the design. Though we have used UPF to show the flow, the methodology is generic and can be incorporated in any other low power specification format.

TABLE I
Simulation Performance Overhead

Design	#Msg	#UPF	#RTL	#Time
Design1	25	43	1K	$< 10\%$
Design2	118	173	200K	$< 10\%$
Design3	1600	366	100K	$< 10\%$

References

[1] Bembaron F., Kakkar S., Mukherjee R. and Srivastava A., 'Low Power Verification Methodology using UPF', In the Proceedings of DVCon, pp. 228-233, 2009.

[2] Crone A. and Chidolue G., 'Functional Verification of Low Power Designs at RTL', In the Workshop for Power And Timing Modeling, Opt. and Sim. (PATMOS), LNCS-4644, pp. 288-299, 2007.

[3] Unified Power Format (UPF 2.0) Standard [Draft Version], IEEE Draft Standard for Design and Verification of Low Power Integrated Circuits, IEEE P1801/D18, 23rd October, 2008.

[4] Hazra A. et al., 'Leveraging UPF-extracted assertions for modeling and formal verification of architectural power intent', In the Proc. of DAC 2010, pp. 773-776.

[5] Synopsys MVRC www.synopsys.com/tools/verification/lowpowerverification/pages/mvrc.aspx

[6] Synopsys MVSIM www.synopsys.com/tools/verification/lowpowerverification/pages/mvsim.aspx

[7] SystemVerilog Language Reference Manual http://www.eda.org/sv/SystemVerilog_3.1a.pdf

[8] Tcl Developer Site http://www.tcl.tk/

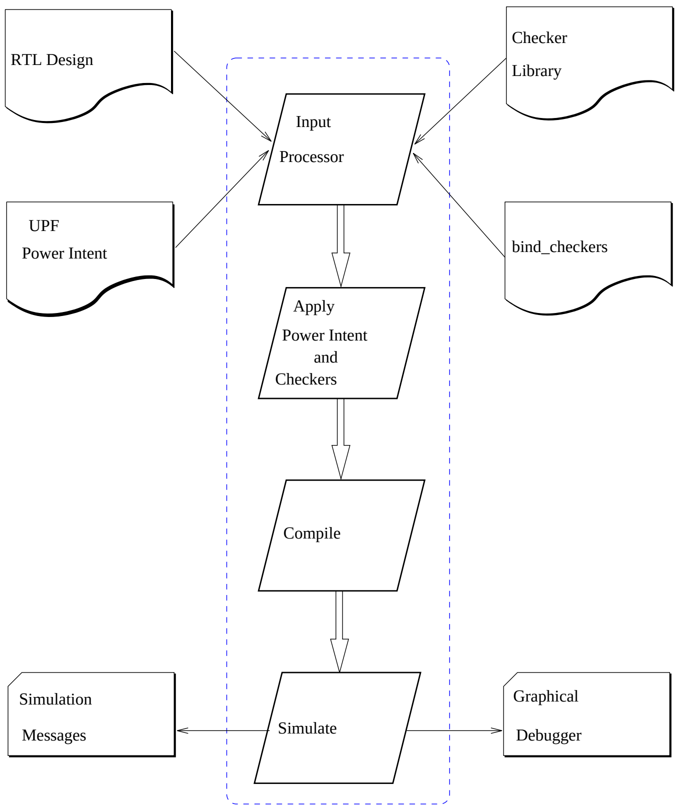

Fig. 4. Architecture of the Tool