INTRODUCTION
The Questa® Power Aware (PA) dynamic simulator (PA-SIM) provides a wide range of automated assertions in the form of dynamic sequence checkers that cover every possible PA dynamic verification scenario. However, design specific PA verification complexities may arise from adoption of one or a multiple of power dissipation reduction techniques, from a multitude of design features – like UPF strategies – as well as from target design implementation objectives. Hence, apart from tool automated checks and PA annotated testbenches, additional and customized PA assertions, checkers, and their monitors are sometimes required to be incorporated in a design.

But a design may already contain plentiful assertions from functional verification parts, often written in SystemVerilog (known as SVA) and bind with the language bind construct. SystemVerilog provides a powerful bind construct that is used to specify one or more instantiations of a module, interface, program, or checker without modifying the code of the target. So, for example, instrumentation code or assertions that are encapsulated in a module, interface, program, or checker can be instantiated in a target module or a module instance in a non-intrusive manner. Still, customized PA checks, assertions, and monitors are often anticipated to be kept separate, not only from the design code but also from functional SVA.

UPF provides a mechanism to separate the binding of such customized PA assertions from both functional SystemVerilog assertions (SVA) and the design. The UPF bind_checker command and its affiliated options allows users to insert checker modules into a design without modifying and interfering with the original design code or introducing functional changes. However, UPF inherits the mechanism for binding the checkers to design instances from the SystemVerilog bind directives. Hence similar to SVA, the UPF bind_checker directive causes one module to be instantiated within another without having to explicitly alter the code of either. This facilitates the complete separation between the design implementation and any associated verification code.

Signals in the target instance are bound by position to inputs in the bind checker module through the port list, exactly the same as in the case for SVA bindings. Thus, the bind 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.

ARTIFACTS OF CUSTOM CHECKERS
The UPF bind_checker syntax and “use model” used to create custom PA assertions for a design and bind the checker through the UPF bind_checker command are shown in detail in the following four successive examples.
Example 1. Syntax of UPF bind_checker:

```
bind_checker <instance_name> \ 
  -module <checker_name> \ 
  -elements <element_list> \ 
  -bind_to module [-arch name] \ 
  -ports {{port_name net_name}[*]} \ 
  -parameters {{param_name param_value}[*]}
```

This UPF command and options are used for inserting checker modules into a design and binding them to the design instances. In the syntax, the `<instance_name>` is the “instance” name (e.g. iso_supply_chk) of the checker module `<checker_name>` (e.g. ISO_SUPPLY_CHECKER). The `-elements <element_list>` is the list of design elements where the checker “instance” will be inserted. The `-module <checker_name>` is the name of a SystemVerilog module for which the verification code is targeted. The verification modules are generally coded in SystemVerilog but bind to either a SystemVerilog or VHDL instance through `-bind_to module [-arch name]`.

Also note that `-ports()` are the association of the design signals to the checker ports. The `<net_name>` argument accepts the symbolic references for signals, power supply ports, supply nets, and supply sets defined in UPF for various UPF strategies. For example, `isolation_signal` or `retention_power_net` can be referenced as follows.

Example 2. `<net_name>` Symbolic Referencing for Various UPF Strategies:

```
<design_scope_name>.<powerdomain_name>
  .<iso_stratgy_name>.isolation_signal
<design_scope_name>.<powerdomain_name>
  .retention_power_net
```

The `-parameters {}` option provides the specification of parameter values on the checker model, where `<param_name>` is the name of the parameter and `<param_value>` is the value of that parameter. If the `-parameters` option is specified, the parameter name of `<param_name>` will be set with the value `<param_value>`. For SystemVerilog, it refers to the parameter; for VHDL it refers to generics. The `<param_value>` is a constant value.

Example 3. A Custom Checker Sample for ISO Control Related Assertion:

```
module ISO_SUPPLY_CHECKER(ISO_CTRL,ISO_PWR,ISO_GND);
import UPF::*;

input ISO_CTRL;
input supply_net_type ISO_PWR;
input supply_net_type ISO_GND;
reg ISO_pg_sig;

assign ISO_pg_sig = get_supply_on_state(ISO_PWR) &&
get_supply_on_state(ISO_GND);

always @(negedge ISO_pg_sig)
  assert(!(ISO_CTRL))
  assert(!$display("At time %0d isolation supply is switched OFF
during isolation period, ISO_CTRL=%b", $time, ISO_CTRL));
endmodule
```
Finally the binding of the ISO control related custom checker, or ISO_SUPPLY_CHECKER, is done as follows.

Example 4. Snippet of UPF Code for Binding the ISO_SUPPLY_CHECKER Checker:

```upf
set_scope /tb/TOP
create_supply_net ISO_PWR
create_supply_net ISO_GND
create_supply_port ISO_PWR_PORT
create_supply_port ISO_GND_PORT
connect_supply_net ISO_PWR_port ISO_PWR_PORT
connect_supply_net ISO_GND_port ISO_GND_PORT
create_supply_set ISO_SS
  -function {power ISO_PWR}
  -function {ground ISO_GND}
create_power_domain PD_mid1
  -supply {primary ISO_SS}
set_isolation iso_PD_mid1
  -domain PD_mid1
  -applies_to_outputs
  -isolation_supply_set ISO_SS
  -location self
  -isolation_signal ctrl

# The ISO_SUPPLY_CHECKER Checker binding in UPF
bind_checker iso_supply_chk
  -module ISO_SUPPLY_CHECKER
  -bind_to mid_vl
  -ports {
    {ISO_CTRL PD_mid1.iso_PD_mid1.isolation_signal}
    {ISO_PWR ./ISO_SS.power}
    {ISO_GND ./ISO_SS.ground}
  }
```

Now, as explained before, the design can remain completely separated from the checker and the binding, as shown in Example 5, shown to the right.

Example 5. Design Completely Separate from Checker and Binding:

```upf
tb();
...
top top(...);
...
endmodule
module top(...);
mid_vl test1_vl(...);
mid_vl test2_vl(...);
mid_vl test3_vl(...);
endmodule;
module mid_vl(...);
...
endmodule
```

Questa PA-SIM accesses the mid_vl module and inserts the (ISO_SUPPLY_CHECKER) checker in all available instances under the \top hierarchical paths of the design, which is defined through the set_scope command in UPF, as shown in Example 4. The combined Examples 3, 4, and 5 explain how to design a PA custom checker; how to bind such a checker in UPF; and how the target design completely separates the checker and its bindings. It is worth noting that the checker sample in Example 3 imports the IEEE standards package import UPF::*, similar to a PA annotated testbench, in order to utilize a different type of function to that of a PA annotated testbench, as shown in Example 6.

Example 6. UPF Import Package Function for Custom Checker:

```upf
get_supply_on_state( supply_net_type arg ) ;
```
This function is actually used for driving and providing the connectivity for supply_net_type ISO_PWR and ISO_GND from the checker module for the design instance mid_vl, which is under the -bind_to command. As shown in Example 2, it is possible to associate an object with a <net_name> present in the active UPF scope. For example, the power domain PD_mid1 in Example 4 is created in the current scope; therefore the isolation control signal PD_mid1.iso_PD_mid1.isolation_signal or the isolation supply set (i.e., power and ground) ./ISO_SS.power and ./ISO_SS.ground are directly accessible by Questa PA-SIM.

CONCLUSION
UPF provides a powerful mechanism to define a custom PA checker or assertion and provides a layer to completely separate it from design code. This is done by embedding the binding of the design and checker within the UPF file through the bind_checker command and its options. As a result, it provides a consolidated verification mechanism and allows Questa PA-SIM to access all instances of a target design with a custom checker within the current scope. The bind_checker assertions are distinctively different from SystemVerilog assertions in that they can access the UPF supply network and other UPF objects. These checkers even work when the power and ground are turned off. However, bind_checker commands and the checkers are exclusively defined and designed for dynamic simulation based verification purposes; hence implementation tools completely ignore the bind_checker command and its associated checkers.

REFERENCES
27 Video Courses Available Covering
- SystemVerilog OOP
- Formal Verification
- Intelligent Testbench Automation
- Metrics in SoC Verification
- Verification Planning
- Introductory, Basic, and Advanced UVM
- Assertion-Based Verification
- FPGA Verification
- Testbench Acceleration
- PowerAware Verification
- Analog Mixed-Signal Verification

UVM and Coverage Online Methodology Cookbooks
Discussion Forum with more than 7850 topics
Verification Patterns Library

www.verificationacademy.com