Probing UPF Dynamic Objects: Methodologies to Build Your Custom Low-Power Verification Platform

Progyna Khondkar

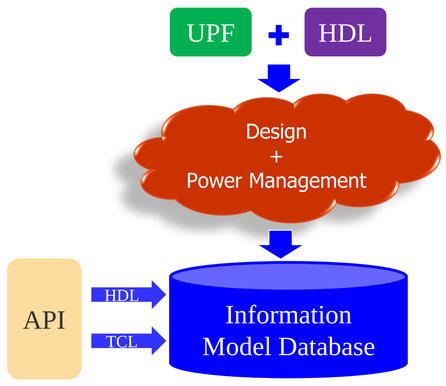
Mentor Graphics, A Siemens Business

accellera SYSTEMS INITIATIVE™

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

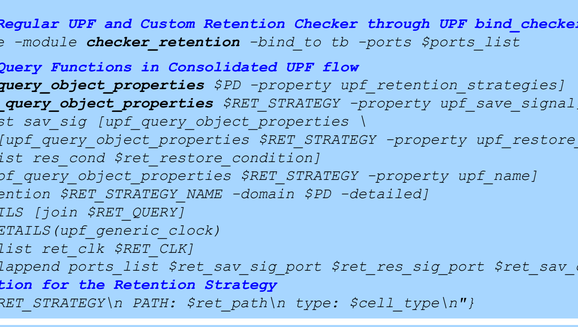
OUTLINE & CONTRIBUTION

- This paper proposes a completely new low-power verification methodology
- On the key concepts of low-power (UPF) information model (UPFIM)
- Allows to query any dynamic properties of UPF objects –
- like continuously probe ON, OFF status of a power domains (during elaboration steps)
- Through Tcl API and passed the objects information on to
- Appropriately instantiated SV API based design codes.
- Example: Tcl API can be used on the simulation execution fly
- To populate any attributes for low-power SystemVerilog checker modules
- Already quarried and bound during elaboration steps into RTL design through bind_checker.



UPF & LOW POWER (TERMINOLOGIES?)

- UPF: Unified Power Format/IEEE 1801- Not just power spec language
- Full set for verification, architecting & implementing low power artifacts on any design
- UPFIM: A database with UPF objects processed in phases
- 5 phases before UPF objects on a design can be quarried
- bind_checker: A mechanism of custom low power checker
- Done by embedding the binding of the design and checker within the UPF/Tcl file
- Provides consolidated verification, messaging & debug environment and
- Simulator access all instances of a target design with a custom checker
- bind_checker assertions are distinctively different from SVA
- They can access all the UPF objects – i.e. UPF power supply, power states etc. through Tcl query or HDL native API available in UPFIM.



RESULTS

- Complete user perception that allows to query UPFIMDB
- On simulation fly & populate any attributes of user low-power SV checker modules
- That are bound in elaboration steps into RTL design through UPF bind_checker.

Phases	UPF Processed on Designs in each Phases
Phase 1	Read UPF specifications
Phase 2	Build UPF model
Phase 3	Recognize the implemented UPF and process simulation controls
Phase 4	Apply UPF model to the design, including any checkers introduced by the UPF **bind_checker**
Phase 5	Query UPF model through Tcl and HDL API based quarry. This phase also implements checkers quarries resulting from the **bind_checker**

- At design optimization (i.e. vopt)
- Simulator leverages consolidated UPF flow
- Consist of UPF commands & UPFIM Tcl API in single UPF file

```
## Custom SV Checker 'ret_checker.sv'
import UPF::*;
module checker_retention(sav_sig, res_sig, sav_cond, res_cond, ret_clk);
    input sav_sig, res_sig;
    input upfExpressionT sav_cond;
    input upfExpressionT res_cond;
    input ret_clk;
    wire clk;
    assign #1step clk = ret_clk;
    property p1;
        @(posedge clk) (sav_sig |-> sav_cond.current_value);
    endproperty
   assert property (p1) $display(">>>%t ----- Save success", $time());
                  else $display(">>>%t ----- Save not executed", $time());
endmodule
```

RESULTS

```
## Regular Power Management UPF 'dut_top.upf'
create_power domaon PD -elements {<list of elements e.g.> top_v1 top_v11}
... ... ...
## Retention Strategy
46 set_retention pd_retention \
47        -domain pd \
52        -save_condition  {!UPF_GENERIC_CLOCK && sc} \
53        -restore_condition {UPF_GENERIC_CLOCK && !UPF_GENERIC_ASYNC_LOAD && rc}
## Regular UPF Commands/Options
## For e.g. defining other PD, PD's supply set, association,
## Power state etc.
## Binding design module, Regular UPF and Custom Retention Checker through UPF bind_checker
bind_checker $instance_name -module checker_retention -bind_to tb -ports $ports_list
```

```
## Utilization of UPF Tcl Query Functions in Consolidated UPF flow
foreach RET_STRATEGY [upf_query_object_properties $PD -property upf_retention_strategies]
 {set ret_save_signal [upf_query_object_properties $RET_STRATEGY -property upf_save_signal]
  set ret_sav_sig_port [list sav_sig [upf_query_object_properties \
set ret_restore_condition [upf_query_object_properties $RET_STRATEGY -property upf_restore_condition]
  set ret_res_cond_port [list res_cond $ret_restore_condition]
  set RET_STRATEGY_NAME [upf_query_object_properties $RET_STRATEGY -property upf_name]
  set RET_QUERY [query_retention $RET_STRATEGY_NAME -domain $PD -detailed]
       array set RET_DETAILS [join $RET_QUERY]
       set RET_CLK $RET_DETAILS(upf_generic_clock)
       set ret_clk_port [list ret_clk $RET_CLK]
       set ports_list {} lappend ports_list $ret_sav_sig_port $ret_res_sig_port $ret_sav_cond_port
## Printing Useful Information for the Retention Strategy
puts "RET INFO STRATEGY: $RET_STRATEGY\n PATH: $ret_path\n type: $cell_type\n"}
```

```
## Useful Information for the Retention Strategy
-- Loading module checker_retention
STRATEGY: /tb/pd.pd_retention1
PATH: /tb/pd.pd_retention1
type: upfRetentionStrategyT
... ... ...
## Assert Property p1 and p2
# ** Error: (vsim-8906) QPA_RET_SEQ_ACT: Time: 36 ns,  clock toggled  during retention period for retention element(s) in
scope '/tb/top_v1': q
# File: ~/test.upf, Line:46, Power Domain:/tb/pd
#                  37 pwr= 1, ret= 1, ret1= 0, clk= 1, rst= 0, d=1, q_v1= 01 01, q_v11= 01
#
# >>>                 37 ----- Save not executed
# >>>                 37 ----- Restore success
```

CONCLUSIONS

- Propose methodology allows to create LP checker
  - User can readily utilize object passing by query function,
    - Hierarchical references as well with or without native SV HDL representations.
- Recognize UPF 3.1 LRM lack clarification on coordinating
  - ***upfExpressionT*** (creates relational objects to captures Boolean expressions for save, restore) with
  - Retention strategy record field or UPF generics, like the UPF_GENERIC_CLOCK, UPF_GENERIC_DATA
  - Specifically UPF mirror object function and ***bind_checker*** for generic requires clarification IEEE 1801.
- Proposed and implemented a novel methodology that paved the way to continuously probe UPF dynamic objects and
  - Allows to build custom low-power verification portfolio on existing low-power simulation platform.
  - If carefully designed, these custom checkers can be reused across any low-power projects.

REFERENCES

[1] Progyna Khondkar, "Low-Power Design and Power-Aware Verification", Hard Cover ISBN: 978-3-319-66618-1, October, 2017, Springer International Publishing.

[2] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801™-2018.

[3] Progyna Khondkar, et al., "How UPF 3.1 Reduces the Complexities of Reusing Power Aware Macros" March, DVCon 2020."

[4] Progyna Khondkar, et al., "Low Power Coverage: The Missing Piece in Dynamic Simulation", February March, DVCon 2018.

[5] Progyna Khondkar, et al., "Free Yourself from the Tyranny of Power State Tables with Incrementally Refinable UPF", February March, DVCon 2017.

[6] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801-2015, 5 December 2015.

Mentor Graphics®

A Siemens Business