2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Low Power Apps
(Shaping the Future of Low Power Verification)

Awashesh Kumar, Mentor, A Siemens Business
(awashesh_kumar@mentor.com)
Madhur Bhargava, Mentor, A Siemens Business
(madhur_bhargava@mentor.com)
Vinay Kumar Singh, Mentor, A Siemens Business
(vinay_singh@mentor.com)
Pankaj Gairola, Mentor, A Siemens Business
(pankaj_gairola@mentor.com),

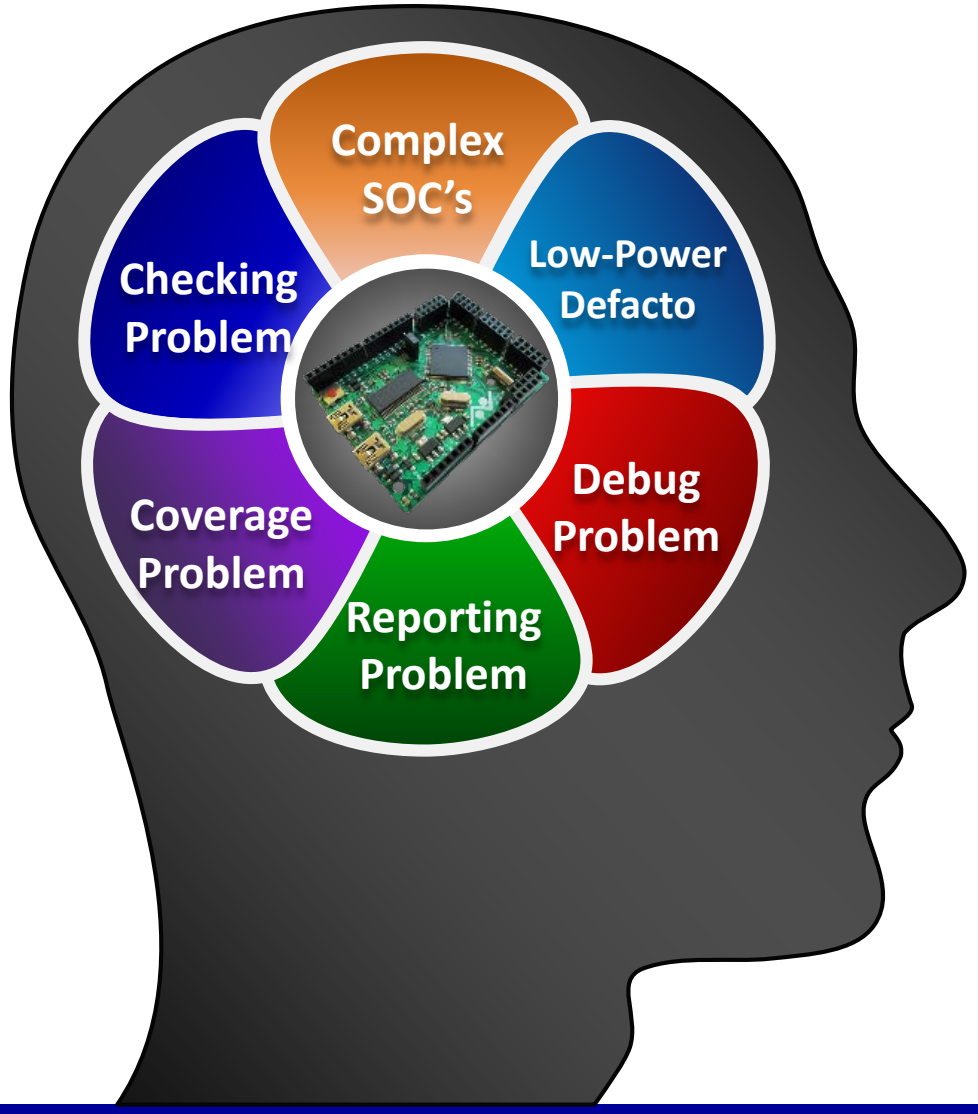
Mentor® A Siemens Business

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- Motivation for paper
- UPF 3.0 Information model
- Low-Power Apps
- UPF 3.0 HDL Package Functions
  - Examples & Case Studies
- UPF 3.0 TCL APIs
  - Example & Case Studies
- Benefits over conventional approaches
- Conclusion

Introduction

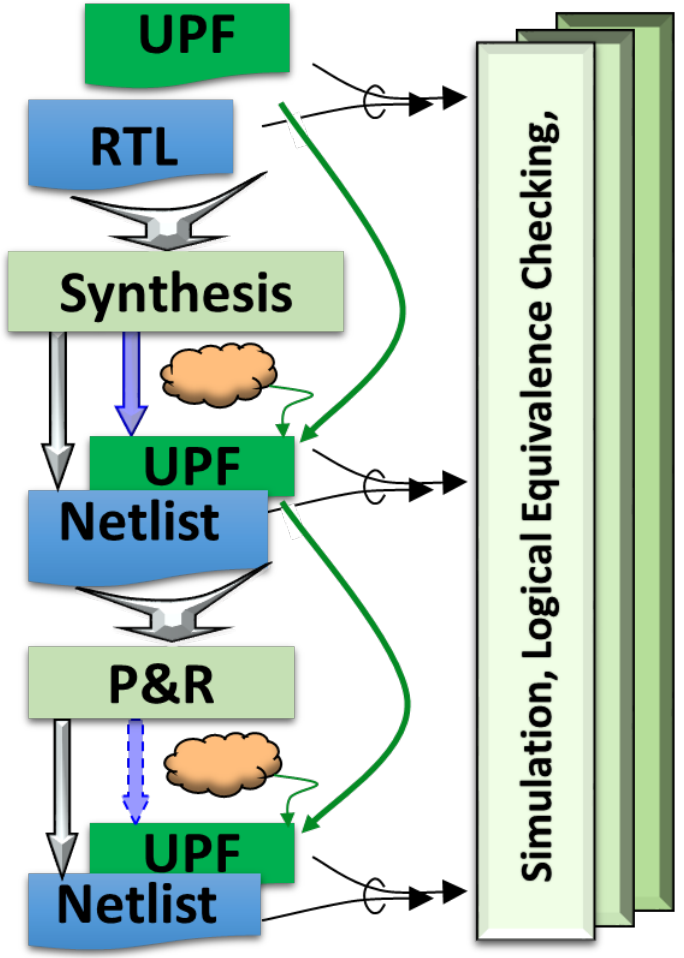


- Complex SOC's
- Low-Power increases complexities
  - Sophisticated power management
- Lot's of time & effort goes into debugging low-power issues
- Low-Power Reports > Too huge
- Verify the power management
  - Coverage
  - Checks

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Unified Power Format (UPF) based Low-Power Verification

- RTL is augmented with a UPF specification
  - To define the power architecture for a given implementation
- RTL + UPF drives implementation tools
  - Synthesis, place & route, etc.
- RTL + UPF also drives power-aware verification
  - Ensures that verification matches implementation



IEEE Std 1801™-2013

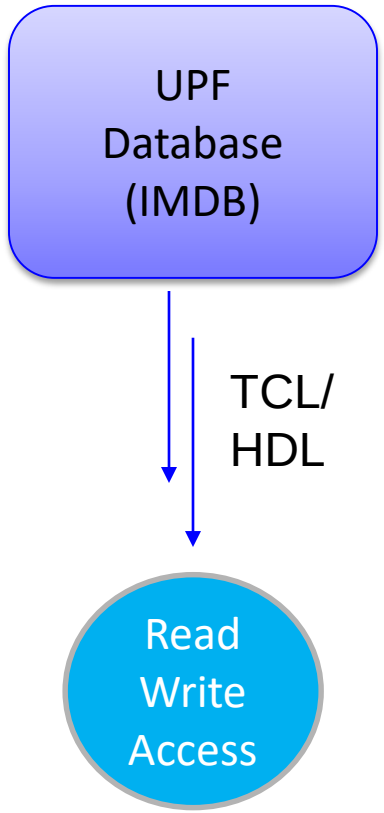
IEEE Std 1801™-2015

Motivation

- Low-Power is now de-facto in the industry and all the designs are power-aware
  - Research says “majority of verification time & effort is spent in debug”
  - Catching low power bugs early is important
    - Intelligent ways of doing coverage & checks
- Problems
  - Designers/Verifications engineers capabilities > Verification tools
  - Users cannot access and manipulate the low-power objects in the same way as they do for RTL
- Need for mechanism to do selective reporting of a part of design is needed

UPF 3.0 Information Model

- Introduced in UPF 1801-2015
- Abstract data model to represents low power objects created in UPF
  - E.g. Power Domain, Power State, Supply Set etc.
- Provides access to properties of low power objects
- API interface; to allow access of objects and properties
  - Tcl Interface:
    - To access objects/properties in a Tcl script or UPF file
  - HDL Interface:
    - to access/manipulate objects/properties in a testbench or simulation model



UPF 3.0 Information Model Cont.

- Native HDL representation
  - For object with dynamic properties e.g. power domain
  - Represented by struct/record in HDL containing two fields
    - A value field – dynamic property value
    - A handle/reference to UPF object – to access other properties of the object

Type Name	SV Representation

upfPdSsObjT	struct {
	  upfHandleT handle;
	  upfPowerStateObjT current_state;
	} upfPdSsObjT

Low-Power Apps

- Low-Power Applications
  - Set of “UPF 3.0 information model HDL package functions and Tcl query functions”
- User can write their own apps and run in simulators
- Innovative ways of writing PA apps
  - Debug, Reporting, Coverage, Checking .. Many more ideas

TCL APIs

- UPF 3.0 information model defines a number of Tcl query command to access the low-power objects
- To get various attributes on a given object
  - `upf_query_object_attributes obj –property <attr_name> -detailed`
- To get the type of the object
  - `upf_query_object_type obj`
- To check if an object belongs to a particular group
  - `upf_object_in_class obj –class <class_id>`
- To get the full hier path of an object relative to given scope
  - `upf_query_object_pathname obj –relative_to <object_handle`

Building Apps with TCL APIs

UPF

set_scope /tb/chip_top

create_power_domain PD_CAMERA -include_scope

create_supply_net pd_pwr -domain PD_CAMERA

create_supply_set ss -function {power pd_pwr} \

-function {ground G_pd_net}

associate_supply_set ss -handle PD_CAMERA.primary

…

Issue:

/tb/chip_top/c shows ‘x’ (corrupted at time 50 ns)

Debug:

Find the source of corruption of this signal

Write a generic app for this
> pa_app x

```
# Step 1: Get the properties of the signal examine
tb/chip_top/c
# 1’bx query tb/chip_top/c
# { {upf_name c} {upf_parent /tb/chip_top} {upf_cell_info #UPFCELL0_71653#}
{upf_port_dir UPF_DIR_OUT} }

# Step 2: Get the properties of cell applied on that signal
query #UPFCELL0_71653#
# {{upf_cell_kind upf_cell_corrupt} {upf_hdl_cell_kind upf_hdlcell_comb} {upf_cell_origin
upf_origin_inferred} {upf_source_extents {#UPFEXTENT2130711#}} }

# Step 3: Get the properties on source extent (extent of power domain, retention strategy
etc.) of the cell
query #UPFEXTENT2130711#
# { {upf_hdl_element tb/chip_top} {upf_object tb/chip_top/PD_CAMERA /*power domain*/} }

# Step 4: Get the supplies of the upf_object (power domain, retention strategy etc.)
query /tb/chip_top/PD_CAMERA -property upf_supply_set_handles
# {/tb/chip_top/PD_CAMERA.primary /tb/chip_top/PD_CAMERA.default_retention
/tb/chip_top/PD_CAMERA.default_isolation}

# Step 5: Get the power (or other relevant function) of the primary supply set
query /tb/chip_top/PD_CAMERA.primary.power
# { {upf_name power} {upf_creation_scope /tb/chip_top/PD_CAMERA} {upf_parent
/tb/chip_top/PD_CAMERA.primary} {upf_ref_kind upf_ref_power} {upf_ref_object
/tb/chip_top/pd_pwr} }

# Step 6: Check the value of UPF supply net
examine tb/chip_top/pd_pwr
# OFF OV
```

Case Studies & Examples

- Tcl Apps – Reporting, Debugging etc..
- **Low-Power App 5: (Debugging App) Trace drivers of UPF objects**

```
proc pa_query_drivers {{object} args} {
set fanin $object
set driver ""
append driver $object
while {[query $fanin -property upf_fanin_conn] != ""} {
            set driver [concat $driver "[examine $fanin] <-"]
            if { [llength [query $fanin -property upf_fanin_conn]] > 1 } {
                        set resolution [query $fanin -property upf_resolve_type]
                        set fanin [query $fanin -property upf_fanin_conn]
                        foreach index $fanin {
                                    set driver [concat $driver "$index [examine $index]"]
                        }
                        set driver [concat $driver "\{$resolution\}"]
                        break
            }
            set driver [concat $driver "[query $fanin -property upf_fanin_conn]"]
            set fanin [query $fanin -property upf_fanin_conn]
    }
    if {[llength $fanin] < 2 } {
            set driver [concat $driver "[examine $fanin]"]
    }
    return $driver
}
```

```
Usage:
pa_query_drivers /tb/t1/m1/b1/vd_bot
Result:
/tb/t1/m1/b1/vd_bot {OFF 0} <-
/tb/t1/m1/b1/vport1_bot {OFF 0}
   <- /tb/t1/m1/vd_mid {OFF 0} <-
/tb/t1/m1/vport1_mid {OFF 0}
      <- /tb/t1/vd_top {OFF 0} <-
/tb/t1/vport2_top {OFF 0}
/tb/t1/vport1_top {OFF 0} {PARALLEL}
```

UPF 3.0 HDL Package Functions

- Provides to access low power object and their properties in HDL
  - Five different classes of HDL functions
- **HDL access functions**: basic functions to access the low power objects and properties
  - `Ex. upfHandleT pd = upf_get_handle_by_name("/top/dut_i/pd")`
- **Immediate read access HDL functions**:
  - `Ex. upfHandleT ps_active_hndl = upf_query_object_properties(ps, UPF_IS_ACTIVE )`
  - `integer ps_on_value = upf_get_value_int(ps_active_hndl)`
- **Immediate write access HDL functions**:
  - `E.g. supply_on(“/tb/dut_i/vdd_net”, 0.9)`
- **Continuous access HDL functions**: enables continuous monitoring of dynamic values
  - `E.g. upfSupplyObjT vdd_monitor = upf_create_object_mirror("/top/dut_i/vdd", "vdd_monitor")`
- **Utility functions**: general utility function to assist users.
  - `E.g. upfClassIdE upf_query_object_type(upfHandleT handle)`

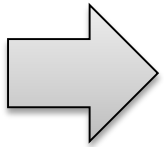
Case Studies & Examples

- HDL Apps – Coverage, Assertions (checking)
- **Low-Power App 1: (Coverage App) Coverage of a low-power design using HDL Package Functions**
  - Coverage app to ensure the full coverage of the simstate property of the primary supply set of all power domains
  - Coverage of “NORMAL-> CORRUPT” and “CORRUPT->NORMAL” transitions for each power domain

Step1: Mirror UPF objects to HDL objects

```
// Native HDL representation for power domains
typedef struct {
          upfHandleT handle;
          upfSimstateT simstates;
} upfPdObjT;

Use the mirror function to continuously monitor the
simstate of all the power domain in the design
    pd_iter = upf_get_all_power_domains();
    pd_hndl = upf_iter_get_next(pd_iter);
    while (pd_hndl) begin
        pd_obj = "power_domain_objs[";
        pd_cnt_str.itoa(pd_cnt);
        pd_obj = {pd_obj, pd_cnt_str};
        pd_obj = {pd_obj, "]"};
        upf_create_object_mirror
(upf_query_object_pathname(pd_hndl), pd_obj);
        pd_cnt++;
        pd_hndl = upf_iter_get_next(pd_iter);
    end
```



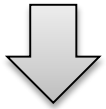
Step 2: Covergroup definition for state and transition coverage

```
    covergroup PD_STATE_COVERAGE (string pd_name, ref
upfSimstateE simstate) @( simstate);
    CORRUPT: coverpoint simstate
                    { bins ACTIVE = {CORRUPT}; }
    NORMAL: coverpoint simstate
                    { bins ACTIVE = {NORMAL}; }
    …
    endgroup

    covergroup PD_TRANS_COVERAGE (string pd_name, ref
upfSimstateE simstate) @( simstate);
    TRANSITION_COVERAGE:coverpoint simstate
    {
           bins OFF_to_ON = (CORRUPT => NORMAL);
           bins ON_to_OFF = (NORMAL => CORRUPT);
    }
    …
    endgroup
```

Step 3: Instantiation of coverage module:

```
    PD_STATE_COVERAGE pd_state_cov [$];
    PD_TRANS_COVERAGE pd_trans_cov [$];
    initial begin
     for (int i = 0; i < pd_cnt; i++) begin
       pd_state_cov[i] = new (upf_query_object_pathname(power_domain_objs[i].handle), power_domain_objs[i].simstate);
       pd_trans_cov[i] = new (upf_query_object_pathname(power_domain_objs[i].handle), power_domain_objs[i].simstate);
     end
    end
```



Coverage Data

Monitor the simstates of a power domain: User can also monitor the simstates of one or more power domains of interest.

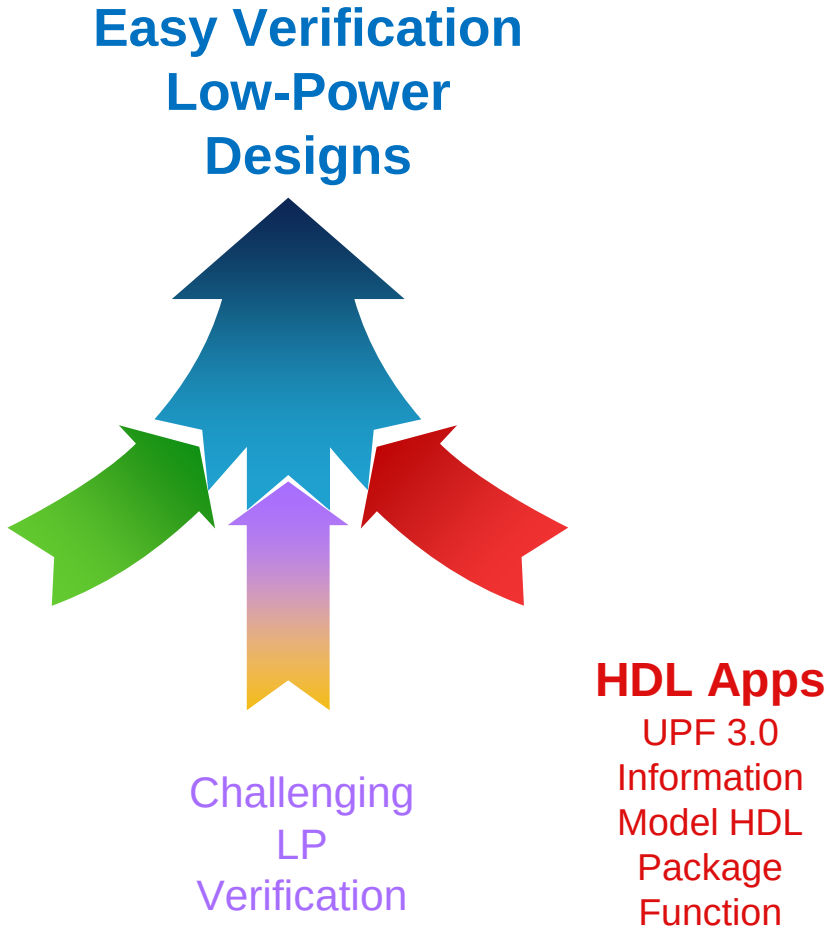
```
always @(power_domain_objs[0].simstate) begin
    $display ($time, "%s Power Domain '%s' simstate changed to '%s'", identstr,
upf_query_object_pathname(power_domain_objs[0].handle), get_simstate_str(power_domain_objs[0].simstate));
end
```

Benefits (Over conventional approaches)

- Achieve early low-power verification closure
- Consistent across tool vendors as it is based on the UPF 3.0 standard
- Flexible & Easy to write low-power apps
- Proposed solution is easily scalable to bigger and more complex design scenarios
- Allows to write PA Apps for
  - Debugging
  - Reporting
  - Self-checking & Coverage

Conclusion

- Low-power designs today are incredibly complex
  - Need of a thorough low-power verification
- Discussed the challenges with the current low-power verification method
- Introduced UPF 3.0 information model
- Low-Power Apps based on UPF 3.0 information model APIs
- Examples & Case studies
  - Consistent, robust and scalable platform.
- Benefits of proposed approach over conventional approaches.



2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

References

- [1] IEEE Std 1801™-2015 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 05 Dec 2015.
- [2] “Amit Srivastava, Awashesh Kumar”, PA-APIs: Looking beyond power intent specification formats, DVCon USA 2015
- [3] “Awashesh Kumar, Madhur Bhargava”, Random Directed Low Power Coverage Methodology: A Smart Approach to Power Aware Verification Closure, DVCon USA 2017
- [4] “Awashesh Kumar, Madhur Bhargava”, Unleashing the Power of UPF 3.0: An innovative approach for faster and robust Low-power coverage, DVCon India 2017

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q&A

Thank You!