2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Low Power Extension in UVM Power Management

Silicon Interfaces®
www.siliconinterfaces.com

Priyanka Gharat (priyanka@siliconinterfaces.com),
Avnita Pal (avnita@siliconinterfaces.com),
Shikhadevi Katheriya (shikha@siliconinterfaces.com)

Abstract- **Incorporating Power Architecture either is in conjunction with or in sequence to functional verification using different languages, many times with different team members using different tools, and divergent approaches leading to potential errors, almost a fourth dimension to our strategy leveraging the test bench architecture. Industry seems to be following a parallel path with respect to Methodologies based test bench Power Architecture, including Unified Power Formats (UPF Both are fundamental requirements to IP and ASIC verification especially in the power saving mobile world. It would be more efficient to do Methodologies based Functional Verification and Coverage interleaved with Low Power Implementation. We have noted previous works in power libraries for VMM and have corrected shortfalls and failings and have modified suitably to work within UVM, which is a much-enhanced Methodology.**

I. Introduction

This Poster demonstrates Power Libraries classes built in System Verilog (UVM_Power) expanding UVM Package Library with Power Domains, Supply Sets, Switches, States and Low Power Strategies as Base Class which may be used within UVM Environment. These Power Base Classes are further built for multi-Cores, Bus Interface, Memory, Etc.

This proposal is to interleave Functional Verification Methodology and Power Architecture in a single existing and widely deployed methodologies-based platform, like UVM.

- With low power strategies, based on UPF and multi-core extensions, a low power or power aware designer or verification engineers would now be able to have a strategy/plan whilst the design/verification is being undertaken.
- As the needs for smaller and Low Power Aware designs needs increase doing the Power Architecture Strategy, especially the Verification as an afterthought post Functional Verification may lead to unwanted re-spins detrimental to costs as well as time to market guidelines.
- Bringing in Power Verification at an earlier stage will bring down the total time for incorporating power strategies resulting in far shorter design cycles.

II. Design And Implementation

An overall UPF structure is created using UVM classes which include different tasks such as creating power domains, different scopes then supply nodes for each of the domains which are created. These classes are used as library and can be extended for creating structures based on DUT/SOC architecture.

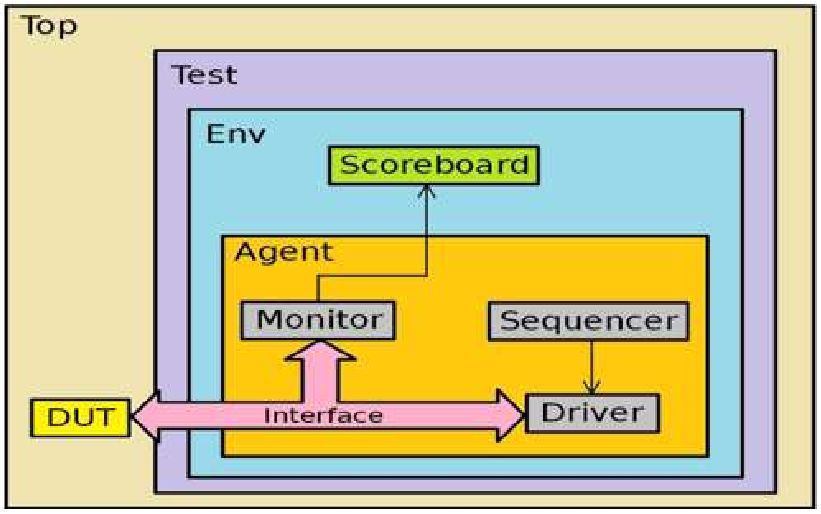


Fig 1. Hierarchical Structure of UVM

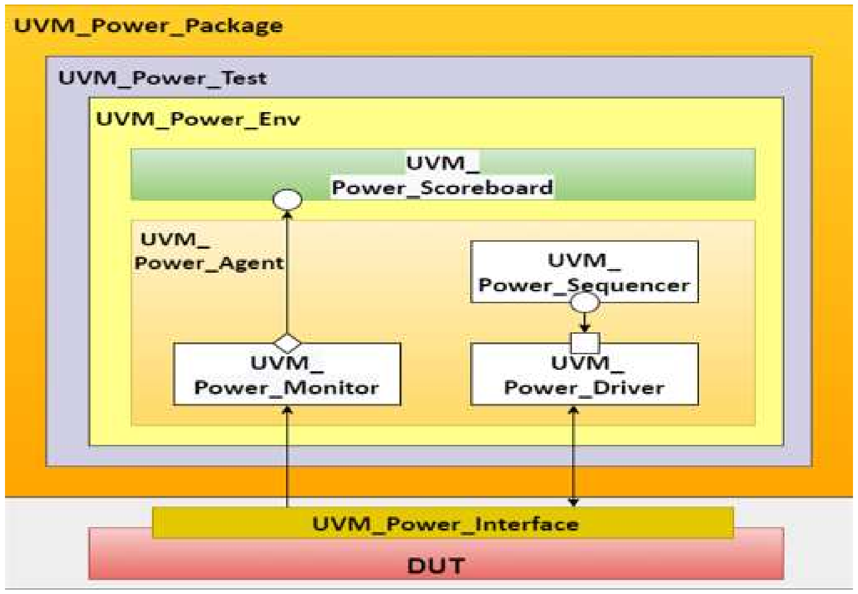


Fig 2. Hierarchical Structure of UVM Power Domain

III. UVM POWER PACKAGE ARCHITECTURE

System Verilog Package declared for Low Power UVM_LP (similarly to power formats used in UPF) and have either a single base class for LP with tasks and functions defined for create_power_domains, create_supply_ports for domains and logic ports for switches, create supply_nets (VDD, VSS for each power-domains) and logic_nets for switches, connect_supply_nets to ports, associate supply_nets, create port & power states & tables, and then for strategies, like level shifters, isolations & retention – logic ports/nets and set the library cells for isolation and retention

Top level consists of base class (UVM_power) and various extended packages for- UVM_power_device, UVM_power_memory, UVM_power_core and further class extend for UVM_power_multicore, UVM_power_ARM, UVM_power_Intel, UVM_power_OPEN_SOURCE extending UVM_power_core.

The base class has several predefined methods (functions and tasks) in the design-

- Power Domains
- Supply sets/nets
- Connect Supply sets to Power Domains.
- Switches - With signals, like Wait_For_Interrupt; Wait_For_Event; Delay_time_for_power_down; Enable_wakeup_timer_interrupt_before_power_down.
- States - normal, standby, connect_standby, retention, sleep, dormant, deepsleep, hibernate, power_down_state; {c0,c1,c2,c3,c4,c5,c6,c7,c8} power_up_state.
- Assign States to Supply sets.
- Strategies for Level Shifters, Retention and Isolation.
- Mapping Strategies to Libraries.
- Virtual functions, tasks & sub-routines for power up and power down, state transitions, etc

The template UVM has UVM_power_test top, UVM_power_Agent, UVM_power_Sequence, UVM_power_Driver, Monitor, Score board, etc..

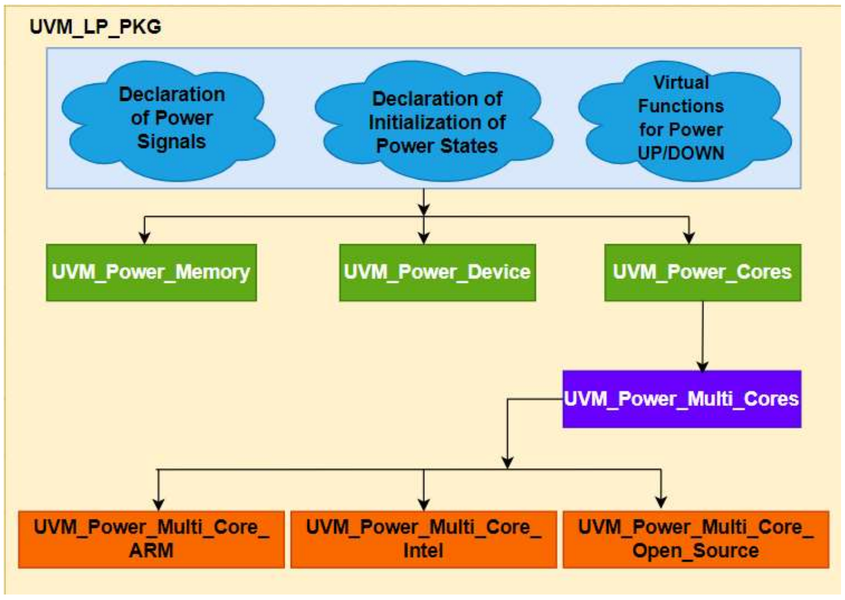

UVM_LP_PKG
Declaration of Power Signals
Declaration of Initialization of Power States
Virtual Functions for Power UP/DOWN
UVM_Power_Memory
UVM_Power_Device
UVM_Power_Cores
UVM_Power_Multi_Cores
UVM_Power_Multi_Core_ARM
UVM_Power_Multi_Core_Intel
UVM_Power_Multi_Core_Open_Source

Fig 3. Flowchart of UVM Low Power Package

IV. Architecture of UVM_Power package for multi-core

Here we are framing an example which is an ARM processor or cluster with single core or it can be formed with multi core processor needed to create a System Verilog encapsulated class with inbuilt task and functions defined in it for Low Power UVM package**.**

The top level consists of base class (ARM_Muli_cluster_UVM_LP_TEST) and various extended packages for- UVM_Power_Agent, UVM_power_multi_cluster_transaction, UVM_power_sequencer, UVM_power_driver. Also there will be several agents for multiple protocols and interfaces like UVM_Power_Agent_PCIe, UVM_Power_Agent_AXI, UVM_Power_Agent_ARM_CHI etc. for large SoC design.

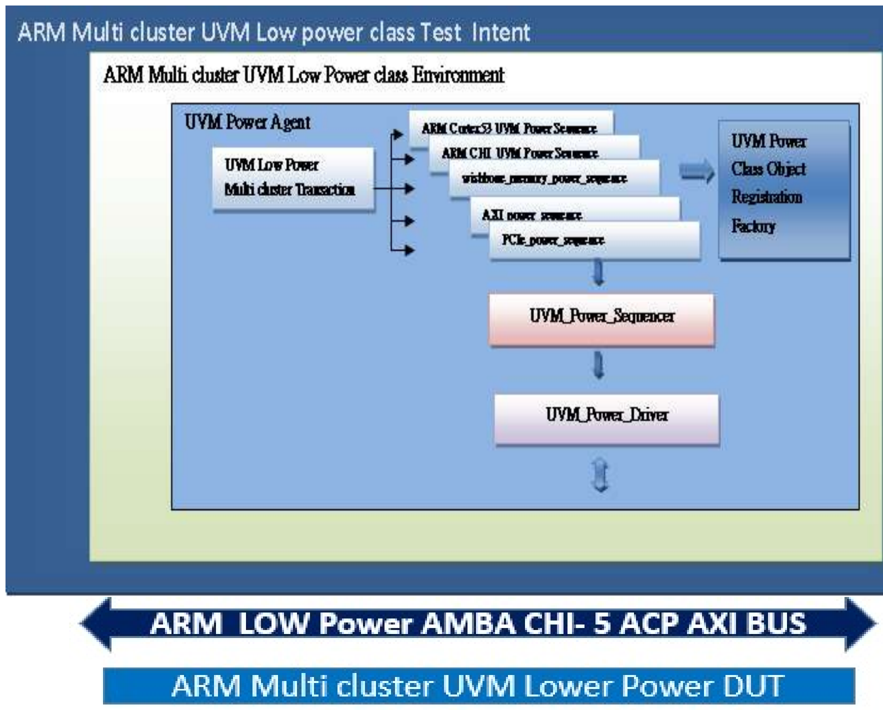

ARM Multi cluster UVM Low power class Test Intent
ARM Multi cluster UVM Low Power class Environment
UVM Power Agent
UVM Low Power Multi cluster Transaction
ARM Cortex53 UVM Power Sequence
ARM CHI UVM Power Sequence
wishbone_memory_power_sequence
AXI power sequence
PCIe_power_sequence
UVM Power Class Object Registration Factory
UVM_Power_Sequencer
UVM_Power_Driver
ARM LOW Power AMBA CHI- 5 ACP AXI BUS
ARM Multi cluster UVM Lower Power DUT

Fig 4. Hierarchical view of UVM Low Power Package for multi core

V. EVIDENCE: INCORPORATING UVM_POWER PACKAGE TO UVM

We are providing the sample code for the UVM Power Package as defined and which may be incorporated in UVM Libraries

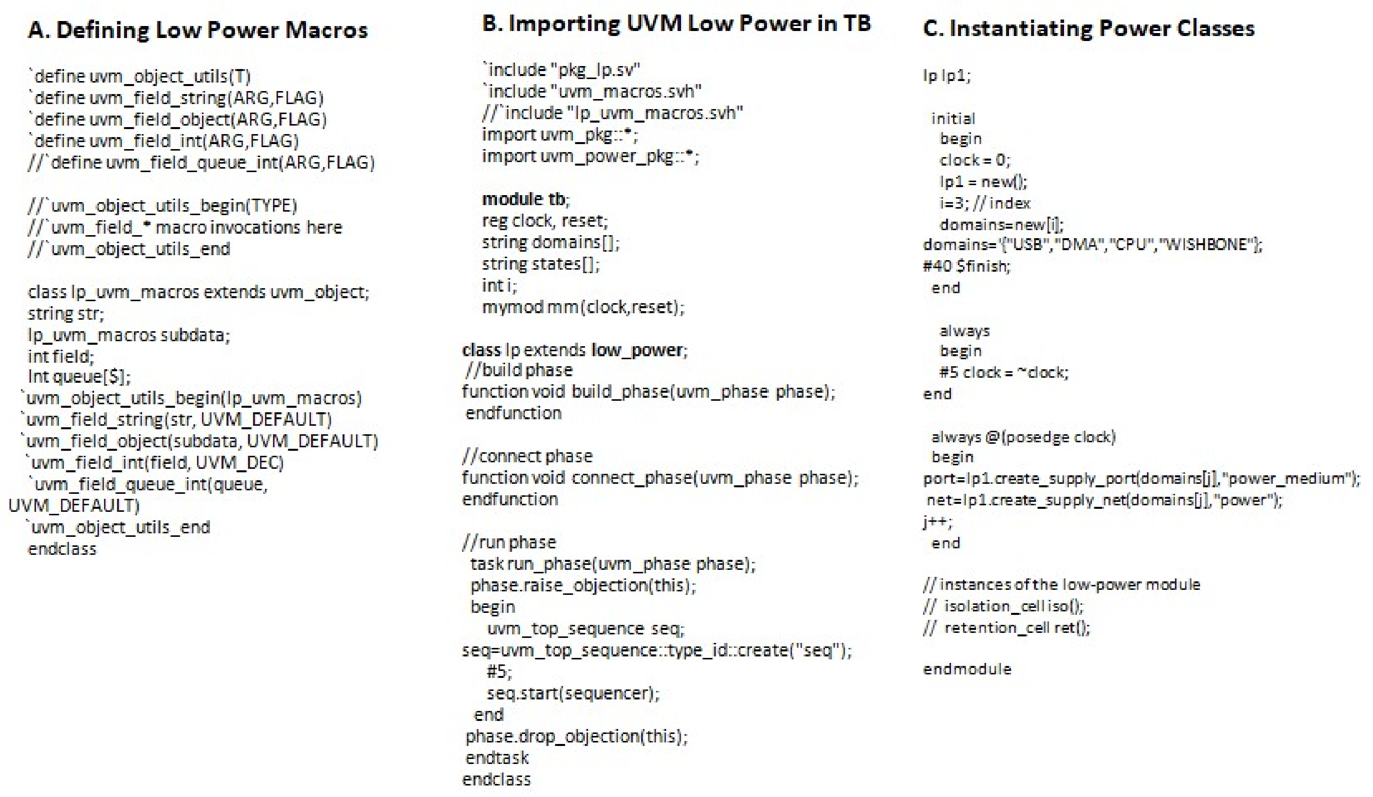
```
package uvm_power_pkg;
    import uvm_pkg::*;
    class uvm_low_power;
    //`uvm_component_utils(uvm_low_power); - Factory Registration
    //function new (string name, uvm_component uvm_low_power);
    //endfunction : new – Construction
        class low_power;
                function string create_power_domain(input string domain_name,input string states, input int
    index);
                        string power_domain[];
                        power_domain=new[index];
                        for(int i=0;i<index;i++)
                        begin
                                power_domain[i] = {"PD_",domain_name}; // domain created here
                                return power_domain[i];
                        end
                endfunction

                function string create_supply_port (input string in_port, input string type_of_signal);
                        if(type_of_signal=="power_high")
                                return {in_port,"_VDDH"};   ...
                        else if(type_of_signal=="power_low")
                                return {in_port,"_VDDL"};
                        else if(type_of_signal=="ground")
                                return {in_port,"_VSS"};
                        else
                                return "NULL";
                endfunction

        function string create_supply_net(input string in_signal, input string type_of_signal);
                if(type_of_signal=="power")
                        return {in_signal,"_Pwr"};
                else if(type_of_signal=="ground")
                        return {in_signal,"_Gnd"};
                else
                return {in_signal,"_net"};
        endfunction

        function string connect_supply_net(input string in_port, input string in_net);
                return {in_port,"_",in_net};
        endfunction

        function bit isolation_cell(input in_signal);
                isolation_cell = in_signal;
        endfunction

        function bit retention_cell(input in_signal,restore);
                reg memory;
                memory = in_signal;
                if(restore == 1) // when restore is enabled then the signal is available at the o/p side.
                retention_cell = memory;
        endfunction
```

```
          function bit power_switch(input in_signal,switch_control);
                    if(switch_control == 1)
                              power_switch = in_signal;
                    else
                              power_switch = 1'bx;
          endfunction
endclass
endpackage
```

V. Evidence: Incorporating Uvm_Power Package To Uvm

As you can see invoking the Low Power Package is as simply as “A” “B” “C” and the good part is that the “A” will come as part of the Low Power Package. In step B, we have included the library package and the header files as well as writing the test bench with instantiating the necessary classes, etc. The Low Power Class can be extended and run the necessary build, connect and run phases.

In Step C here we are passing a dynamic array of the top level modules present in the DUT which would need to have Power Domains. These are typically already instantiated in the test bench top module and can be extracted using an external python script. The calls to the functions will return the value as required in our Low Power domain.



A. Defining Low Power Macros

```
 `define uvm_object_utils(T)
 `define uvm_field_string(ARG,FLAG)
 `define uvm_field_object(ARG,FLAG)
 `define uvm_field_int(ARG,FLAG)
 //`define uvm_field_queue_int(ARG,FLAG)

 //`uvm_object_utils_begin(TYPE)
 //`uvm_field_* macro invocations here
 //`uvm_object_utils_end

 class lp_uvm_macros extends uvm_object;
 string str;
 lp_uvm_macros subdata;
 int field;
 Int queue[$];
`uvm_object_utils_begin(lp_uvm_macros)
`uvm_field_string(str, UVM_DEFAULT)
`uvm_field_object(subdata, UVM_DEFAULT)
 `uvm_field_int(field, UVM_DEC)
 `uvm_field_queue_int(queue,
UVM_DEFAULT)
 `uvm_object_utils_end
 endclass
```

B. Importing UVM Low Power in TB

```
  `include "pkg_lp.sv"
  `include "uvm_macros.svh"
  //`include "lp_uvm_macros.svh"
  import uvm_pkg::*;
  import uvm_power_pkg::*;

  module tb;
  reg clock, reset;
  string domains[];
  string states[];
  int i;
  mymod mm(clock,reset);

class lp extends low_power;
 //build phase
function void build_phase(uvm_phase phase);
 endfunction

//connect phase
function void connect_phase(uvm_phase phase);
endfunction

//run phase
 task run_phase(uvm_phase phase);
 phase.raise_objection(this);
 begin
   uvm_top_sequence seq;
seq=uvm_top_sequence::type_id::create("seq");
   #5;
   seq.start(sequencer);
  end
 phase.drop_objection(this);
 endtask
endclass
```

C. Instantiating Power Classes

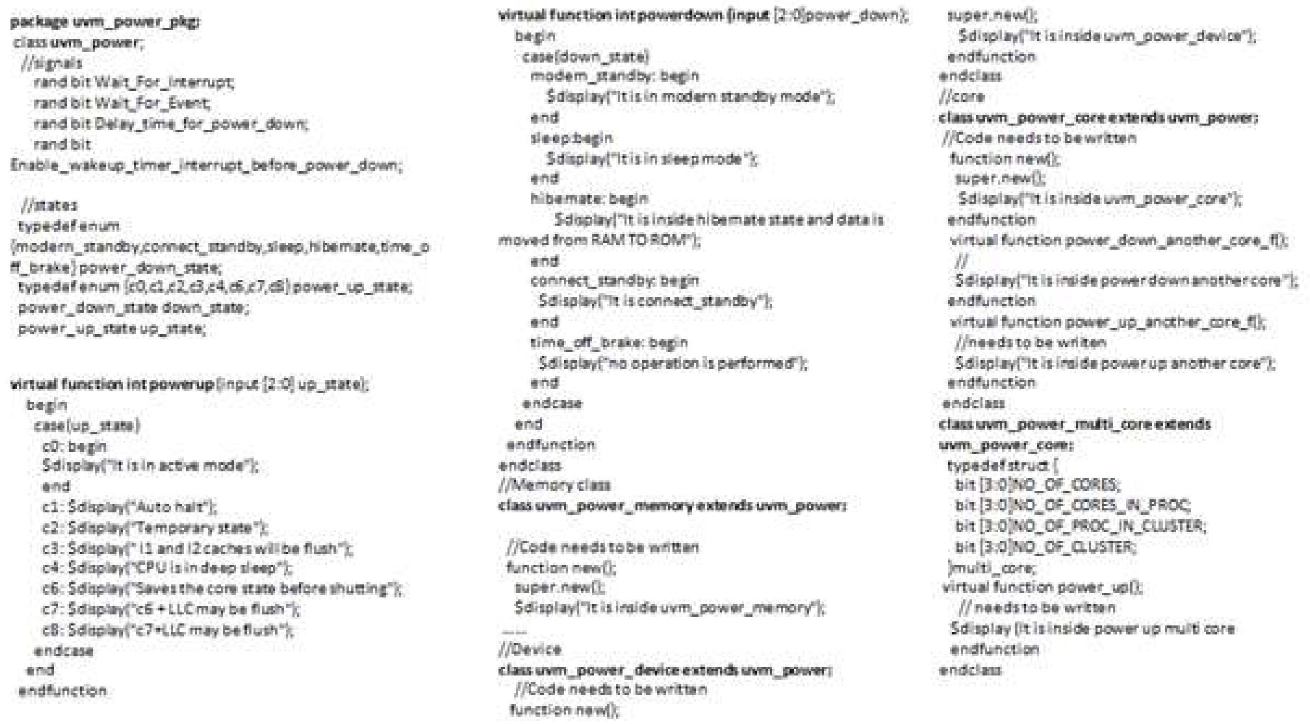
```
lp lp1;

 initial
  begin
  clock = 0;
  lp1 = new();
  i=3; // index
  domains=new[i];
domains='{"USB","DMA","CPU","WISHBONE"};
#40 $finish;
 end

  always
  begin
  #5 clock = ~clock;
end

 always @(posedge clock)
 begin
port=lp1.create_supply_port(domains[j],"power_medium");
 net=lp1.create_supply_net(domains[j],"power");
j++;
 end

// instances of the low-power module
//  isolation_cell iso();
//  retention_cell ret();

endmodule
```

VI. Evidence: Extending UVM_Power Package For Multi-core

As given in IV, this is code incorporating the UVM Power Libraries and extending the same to multi-core, particularly for powerup and powerdown.

```
class my_power extends uvm_power;

 //`uvm_component_utils(my_power)   Factory
Registeration

  //Constructor
  function new(string name = "", uvm_component
parent);
    super.new(name,parent);
  endfunction
uvm_power power;

 initial begin
power = new();

// down_state =
uvm_power_pkg::uvm_power::c1;
power.powerup(2);
power.powerdown(3);

power.sequential_power_down_up_multi_core_f();
power.power_up_another_core_f();

end
```

```
//build phase
 function void build_phase(uvm_phase phase);

 endfunction

 //connect phase
 function void connect_phase(uvm_phase phase);

 endfunction

 //run phase
 task run_phase(uvm_phase phase);
   phase.raise_objection(this);
   begin
     uvm_top_sequence seq;
     seq =
uvm_top_sequence::type_id::create("seq");
     #5;
     seq.start(sequencer);
   end
   phase.drop_objection(this);
 endtask
endclass
```

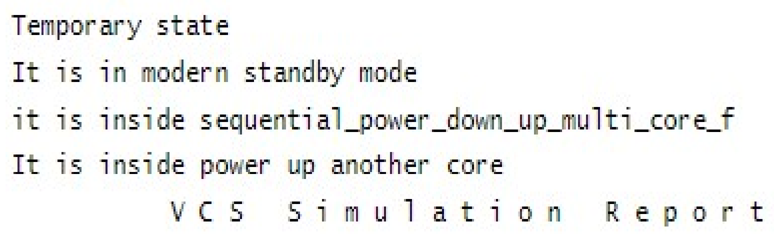


```
package uvm_power_pkg;
 class uvm_power;
  //signals
    rand bit Wait_For_Interrupt;
    rand bit Wait_For_Event;
    rand bit Delay_time_for_power_down;
    rand bit
Enable_wakeup_timer_interrupt_before_power_down;

  //states
  typedef enum
{modern_standby,connect_standby,sleep,hibernate,time_o
ff_brake} power_down_state;
  typedef enum {c0,c1,c2,c3,c4,c6,c7,c8} power_up_state;
  power_down_state down_state;
  power_up_state up_state;


virtual function int powerup(input [2:0] up_state);
   begin
    case(up_state)
     c0: begin
     $display("It is in active mode");
     end
     c1: $display("Auto halt");
     c2: $display("Temporary state");
     c3: $display(" l1 and l2 caches will be flush");
     c4: $display("CPU is in deep sleep");
     c6: $display("Saves the core state before shutting");
     c7: $display("c6 + LLC may be flush");
     c8: $display("c7+LLC may be flush");
    endcase
   end
  endfunction
```

```
virtual function int powerdown (input [2:0]power_down);
   begin
    case(down_state)
     modem_standby: begin
       $display("It is in modern standby mode");
     end
     sleep:begin
       $display("It is in sleep mode");
     end
     hibernate: begin
        $display("It is inside hibernate state and data is
moved from RAM TO ROM");
     end
     connect_standby: begin
      $display("It is connect_standby");
     end
     time_off_brake: begin
      $display("no operation is performed");
     end
    endcase
   end
  endfunction
endclass
//Memory class
class uvm_power_memory extends uvm_power;

  //Code needs to be written
  function new();
   super.new();
   $display("It is inside uvm_power_memory");
.....
//Device
class uvm_power_device extends uvm_power;
   //Code needs to be written
  function new();
```

```
   super.new();
    $display("It is inside uvm_power_device");
   endfunction
 endclass
 //core
 class uvm_power_core extends uvm_power;
  //Code needs to be written
   function new();
    super.new();
     $display("It is inside uvm_power_core");
   endfunction
   virtual function power_down_another_core_f();
    //
    $display("It is inside power down another core");
   endfunction
   virtual function power_up_another_core_f();
    //needs to be writen
    $display("It is inside power up another core");
   endfunction
  endclass
 class uvm_power_multi_core extends
 uvm_power_core;
   typedef struct {
    bit [3:0]NO_OF_CORES;
    bit [3:0]NO_OF_CORES_IN_PROC;
    bit [3:0]NO_OF_PROC_IN_CLUSTER;
    bit [3:0]NO_OF_CLUSTER;
   }multi_core;
  virtual function power_up();
     // needs to be written
    $display (It is inside power up multi core
    endfunction
 endclass
```

RESULT



```
Temporary state
It is in modern standby mode
it is inside sequential_power_down_up_multi_core_f
It is inside power up another core
          V C S   S i m u l a t i o n   R e p o r t
```

CONCLUSION

Incorporating Power Management architecture within UVM methodologies alleviates challenges of functional verification engineer and power management divide. We proposed in-built Power Domain Classes as extension to UVM Package as the Library may be extended to Devices, multi-Cores, Memories, Bus Interface, etc. giving one package for implementation ease. Consolidation of Functional Verification and Power Management will lead to reduced verification time and better chance to meet the time to market deadlines.

REFERENCES

[1] UVM Community (accellera.org) https://accellera.org/community/uvm

[2] Guide to changes in IEEE 1801-2013 (UPF 2.1) (techdesignforums.com)

[3] Verification Methodology Manual for Low Power https://www.synopsys.com/company/resources/synopsys-press/vmm-low-power.html