2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Integrating L1&L2 Cache for multi-Core UVM based extended Low Power Library Package

Avnita Pal, *Silicon Interfaces*
Priyanka Gharat, *Silicon Interfaces*
Sastry Puranapanda, *Silicon Interfaces*

Silicon Interfaces®
a software and vlsi design center

accellera SYSTEMS INITIATIVE

Introduction

- In DAC 2022 and DVCon 2022 India “Low Power extended UVM Power Management” paper.
  - Showed the Low Power UVM Package based on UPF constructs.

Good Reference paper: Low Power Extension In UVM Power Management
https://youtu.be/kkdvOSXxMRc

- This presentation shows past work now updated as UPVM™
  - UPVM Standard for Low Power Validation Class.
    - Extends in UVM Scoreboard to power strategies and mapping.
- Further, the presentation shows the extension of Low Power UVM Libraries to multi-Core with L1 & L2 caches.
  - To incorporate multi-Core Low Power Libraries.
  - Deploying theses Libraries in existing UVM Architecture.

Problem Statement

- Suboptimal approach: Current integration of Power Architecture after or in pipeline to Functional Verification.

- Possible Solution: Previous works demonstrated the benefits of a Unified Platform like UVM for developing Library Components, encompassing Low Power Strategies, Functional Verification Methodology, and UPF-based Low Power Architecture for Devices.

- Efficiency Enhancement: Optimal strategy requires integrating Power Architecture from the beginning with Methodologies based Functional Verification and Coverage, and Low Power Implementation for Devices, Memory, Interface, multi-Cores with L1 & L2.

- Industry parallelism in:
  - Methodologies based test bench
  - Power Architecture (including UPF)

- Fundamental requirements for IP and ASIC verification, particularly in power-saving mobile applications.

- Power Architecture often treated as an afterthought post Functional Verification, adding a fourth dimension to the test bench strategy.

- Leading EDA tools providers offer:
  - Functional Simulation with UVM switches
  - Separate Strategies for Low Power and Power Awareness.

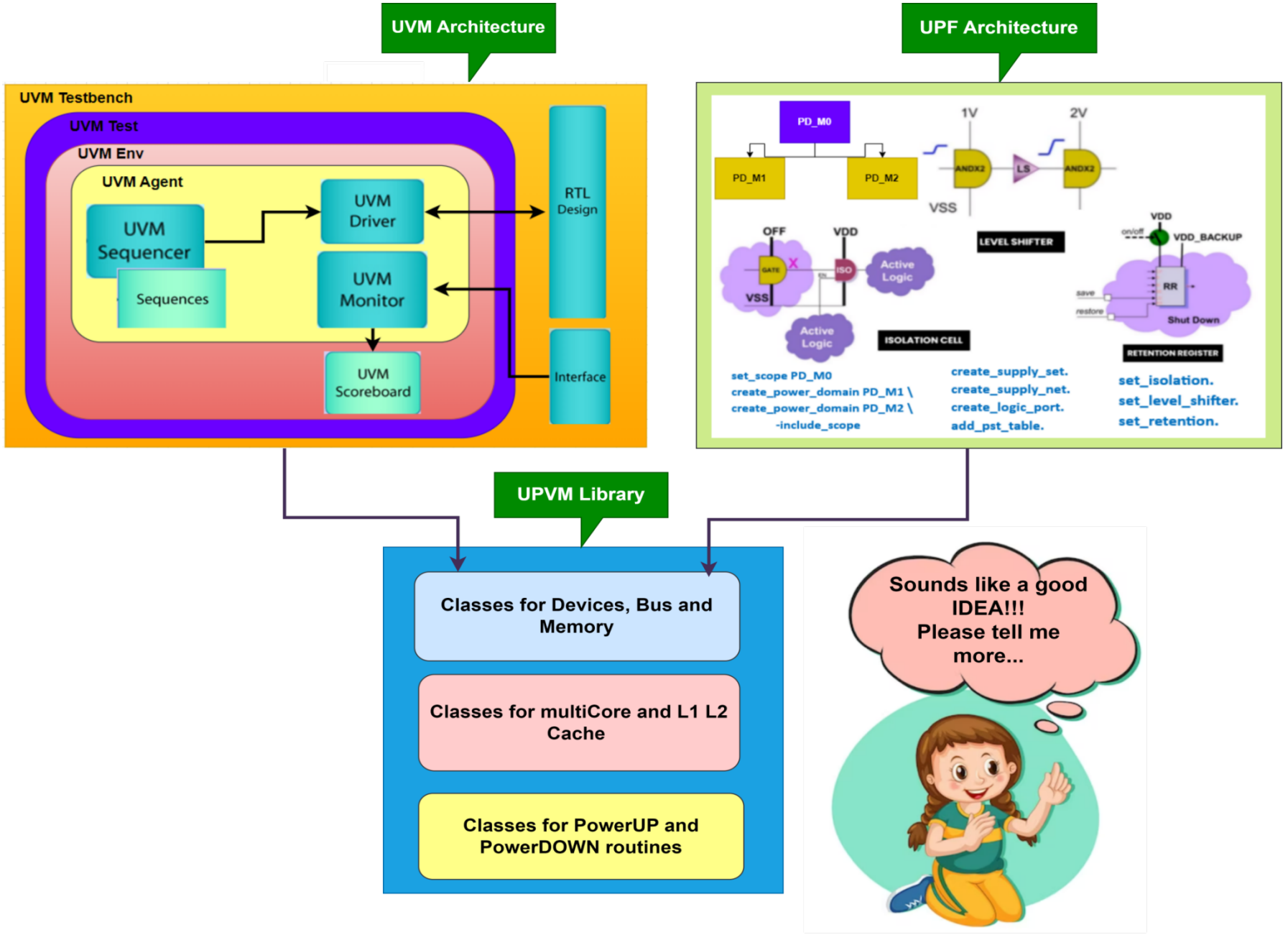
- We propose enhance efficiency by combining Methodologies based Functional Verification, Coverage, and Low Power Implementation.

Main Idea

- Expanding UVM-based Object Classes to encompass Low Power for various components (devices, memory, bus interfaces, cores and cache) enables enhanced reusability and construction within the UVM Environment.

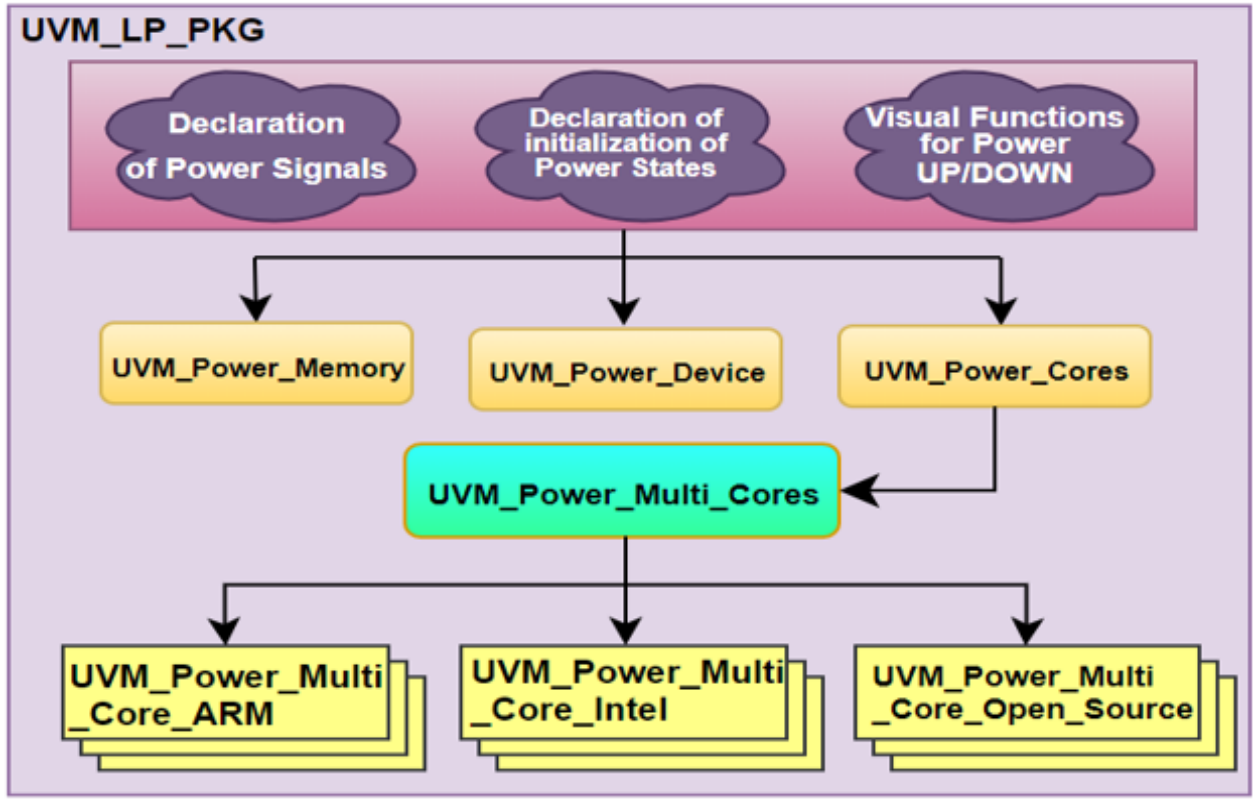
- Incorporating low-power principles and providing API calls for managing multi-core operations and transitioning between states used in processor.
  - Leveraging SV Classes along with DPI connecting with ASM for Low Power routines of L1 & L2 cache verification, promoting low power management, effective design testing using verification methodologies.

UVM Power Library Package, now UPVM



UVM Low Power Package Architecture

- Building upon prior research, this slide focuses on extending the power management structure for multi-Core devices.
- The Power Management structure is dependent on the Power state of each domain, which triggers various virtual functions, tasks, and sub-routines.
- This approach ensures that the Power Management structure is fully integrated into the verification process, providing efficient and effective power management for the device under test.

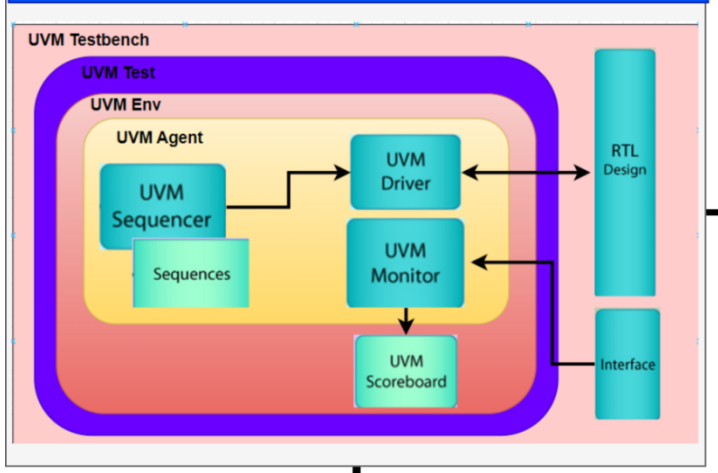


UVM - for Devices and Memory

- UVM Class Libraries: Utilized for code management & upf_version.
- Power Domain tagging extension to ***create_power_domain***.
- Voltage Support: Extending ***create_supply_port***, ***create_supply_net***, ***connect_supply_net***, ***set_domain_supply_net***.
- Power State Management: As ***create_power_switch***, ***add_port_state***, ***create_pst, add_pst_state***.
- Level Shifter Insertion: ***set_level_shifter***.
- Retention and Isolation: Derived from ***set_retention***, ***set_retention_control***, ***set_isolation***, ***set_isolation_control***.
- Mapping functions

Hierarchy Structure of Low Power

UVM Architecture



UVM Library

```
class uvm_test;
`uvm_component_utils
endclass

class uvm_env;
`uvm_component_utils
endclass

class uvm_agent;
`uvm_component_utils
endclass
...
...
...
class uvm_sequence_item;
`uvm_component_utils
endclass
```

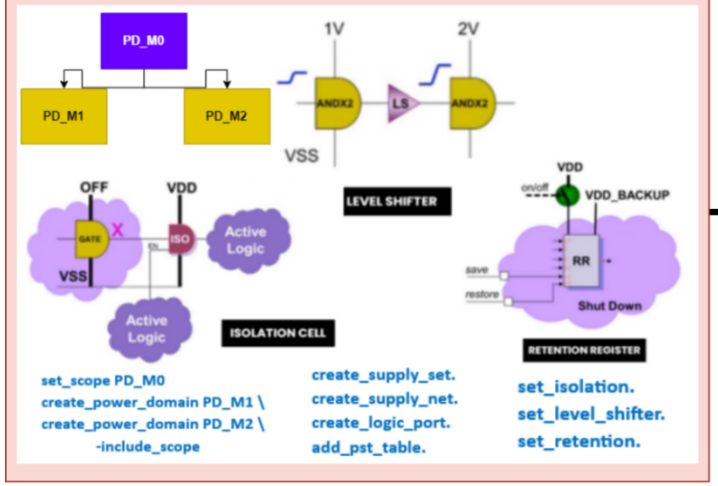
My UVM Testbench

```
class my_test extends uvm_test;
`uvm_component_utils(my_test)
endclass

class my_env extends uvm_env
`uvm_component_utils(my_env)
endclass

class my_agent extends uvm_agent
`uvm_component_utils(my_agent)
endclass
...
...
...
class my_seq_item extends
uvm_sequence_item
`uvm_object_utils(my_seq_item)
endclass
```

UPF Architecture



UPF UVM Library

```
class uvm_lp_create_power_domain;
      function void create_power_domain();
      function void set_design_top();
      function void set_scope();
      function string create_supply_port();
      function string create_supply_net();
      function void set_domain_supply_net();
      function string connect_supply_net();
endclass

class uvm_lp_create_power_switch;

class uvm_lp_create_isolation_logic;

class uvm_lp_create_retention_logic;

class  uvm_lp_create_level_shifter_logic;
....
...
```

My UPVM Testbench

```
class my_domain extends class uvm_lp_create_power_domain;
`uvm_component_utils(my_domain)
endclass

class my_power_switch extends uvm_lp_create_power_switch;
`uvm_component_utils(my_power_switch)
endclass

class my_iso_strat extends uvm_lp_create_isolation_logic;
`uvm_component_utils(my_iso_strat)
endclass

class my_ret_strat extends uvm_lp_create_retention_logic;
`uvm_component_utils(my_ret_strat)
endclass

class my_L_shift_strat extends uvm_lp_create_level_shifter_logic;
`uvm_component_utils(my_L_shift_strat)
endclass
```

Original: Low Power extensions to UVM

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_uv
m_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

function void connect_phase(uvm_phase phase);
endfunction

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
endmodule
```

C. Instantiating Power Classes

```
module top;
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
```

Original: Extending UVM Package to multi-Core

```
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_c
ore_f();
power.power_up_another_core_f();
end
```

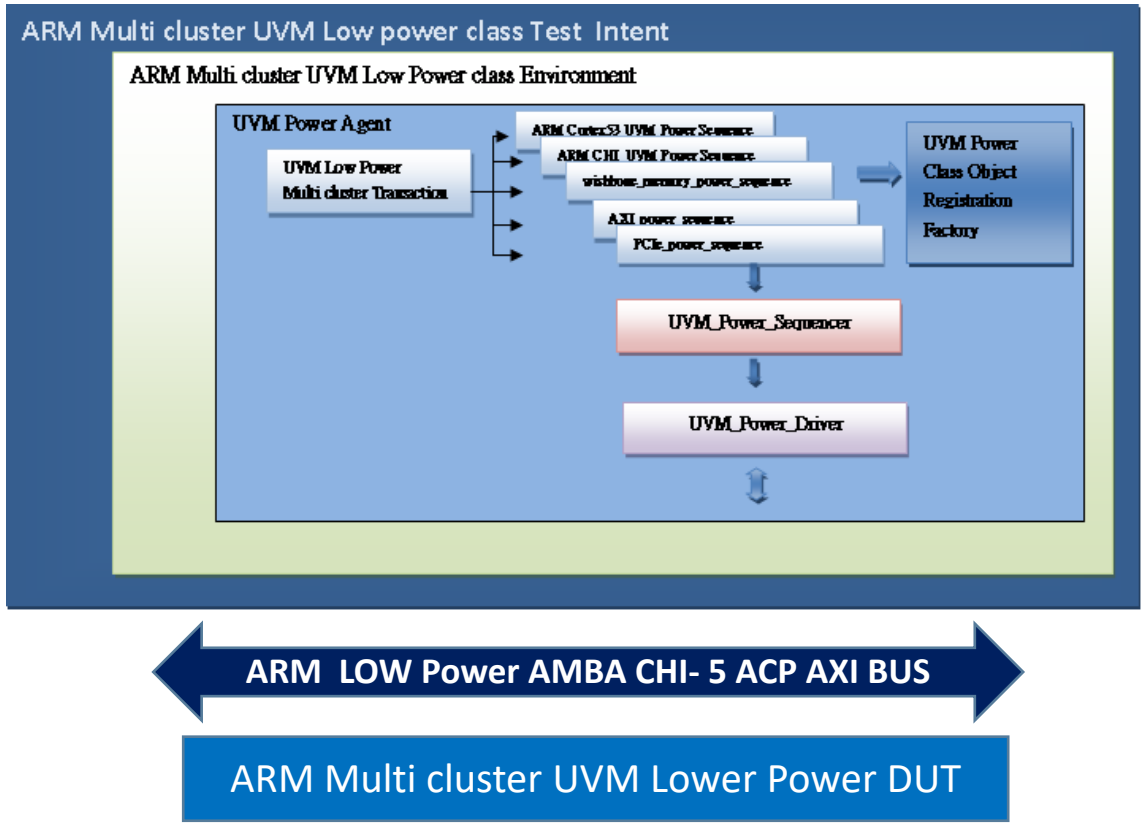


Fig. ARM Multi cluster UVM Low Power class test intent Architecture

Current: A. UVM Low Power DPI Package

```
//FIRST SOURCE CODE

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "svdpi.h"

//#include<ARMv6T2.h>

typedef enum {CLEAN_BY_SETWAY,
            INVALIDATE_BY_SETWAY,
            CLEAN_INVALIDATE_BY_SETWAY,
            CLEAN_BY_VA_TO_POC,
            ---
      }cmo_type_e;

typedef enum {wfi, not_of_wfi,
            wfe,not_of_wfe,
            standbywfi,not_of_standbywfi,
            standbywfe,not_of_standbywfe
      }power_standby_methods_e;

//***Powerdown
//1st step
void disable_cache_func() {
        asm volatile (
            "mrs x0, SCTLR_EL3\n\t"
            "bic x0, x0, #(1 << 12)\n\t"
            "bic x0, x0, #(1 << 2)\n\t"
            "msr SCTLR_EL3, x0\n\t"
            "isb"
            );
}
```

```
//2nd step
#define CLEAN_INVALIDATE_DCACHE_MACRO(op) ({\
      asm("dmb  ish");                    /* ensure all prior inner-shareable accesses have been observed*/\
      asm("mrs  x0, CLIDR_EL1"); \
      asm("and  w3, w0, #0x07000000");    /* get 2 x level of coherence*/\
      asm("lsr  w3, w3, #23"); \
      asm("cbz  w3, "#op"_finished"); \
      asm("mov  w10, #0");                /* w10 = 2 x cache level*/\
      asm("mov  w8, #1");                 /* w8 = constant 0b1*/\
      asm(#op"_loop_level:"); \
      asm("add  w2, w10, w10, lsr #1");   /* calculate 3 x cache level*/\
      asm("lsr  w1, w0, w2");             /* extract 3-bit cache type for this level*/\
      asm("and  w1, w1, #0x7"); \
      asm("cmp  w1, #2"); \
      asm("b.lt "#op"_next_level");       /* no data or unified cache at this level*/\
      asm("msr  CSSELR_EL1, x10");        /* select this cache level*/\
      asm("isb");                         /* synchronize change of csselr*/\
      asm("mrs  x1, CCSIDR_EL1");         /* read ccsidr*/\
      asm("and  w2, w1, #7");             /* w2 = log2(linelen)-4*/\
      asm("add  w2, w2, #4");             /* w2 = log2(linelen)*/\
      asm("ubfx w4, w1, #3, #10");        /* w4 = max way number, right aligned*/\
      asm("clz  w5, w4");                 /* w5 = 32-log2(ways), bit position of way in dc operand*/\
      asm("lsl  w9, w4, w5");             /* w9 = max way number, aligned to position in dc operand*/\
      asm("lsl  w16, w8, w5");            /* w16 = amount to decrement way number per iteration*/\
      asm(#op"_loop_way:"); \
```

Current: UVM Low power DPI package:

- The C code shown in Section A is being called inside uvm_lp_core_pd_pkg package using import “DPI-C” keyword. This helps in connecting ARM routines defined using C assembler language in SV.

```
include "arm_cortex_a53_assembly_code.c"
package uvm_lp_core_pd_pkg;
        ........

        uvm_lp_core_power_standby_methods_e wf;
        uvm_lp_core_cmo_type_e cmo_type;
        uvm_lp_core_barrier_type_e barrier;

        import "DPI-C" function void disable_cache_func();
        import "DPI-C" function void clean_invalidate_dcache_func(cmo_type);
        import "DPI-C" function void cpu_extended_contrl_reg_func();
        import "DPI-C" function void barrier_func(barrier);
        import "DPI-C" function void transition_func(wf);
        import "DPI-C" function void debug_sig_func(bit DBGPWRDUP);
        import "DPI-C" function void activate_output_clamp_func(bit CLAMPCOREOUT);
        import "DPI-C" function void cpu_processor_power_func(bit nCPUPORESET);
        import "DPI-C" function void power_domain_cpu_func(bit PDCPU);
endpackage
```

Current: Functional Description of Power Domains for PowerDown

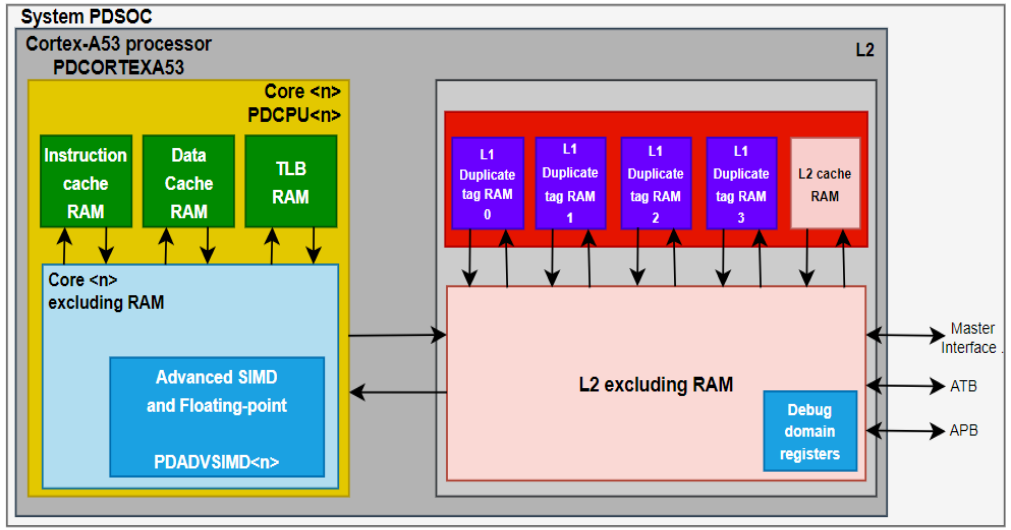


Fig 3. ARM Cortex A53 Power Domain Block Diagram

```
import uvm_lp_core_pd_pkg::*;

class uvm_power_core extends uvm_power;

        //This is the factory registration for low power uvm
        `uvm_lp_component_utils(uvm_power_core)

        function new();
                super.new();
        endfunction

        virtual task uvm_lp_disable_cache_core;
                disable_cache_func();
        endtask

        virtual task uvm_lp_clean_invalidate_dcache_core;
                clean_invalidate_dcache_func(CLEAN_BY_SETWAY);
        endtask
endclass
```

```
class uvm_power_multicore extends uvm_power_core;

        //This is the factory registration for low power uvm
        `uvm_lp_component_utils(uvm_power_multicore)

        typedef struct {
                bit [3:0]NO_OF_CORES;
                bit [3:0]NO_OF_CORES_IN_CLUSTER;
                bit [3:0]NO_OF_CLUSTER;
                bit [3:0]NO_OF_CORES_IN_PROC;
        }multi_core;

        function new();
                super.new();
        endfunction

        virtual task core_power_down;
                begin
                uvm_lp_disable_cache_core();
                uvm_lp_clean_invalidate_dcache_core();
                uvm_lp_cpu_extended_control_reg_core();
                uvm_lp_barrier_core();
                uvm_lp_transition_core();
                uvm_lp_debug_sig_core();
                uvm_lp_activate_output_clamp_core();
                uvm_lp_cpu_processor_power_core();
                uvm_lp_power_domain_cpu_core();
                end
        endtask
endclass
```

```
module tb;
        import uvm_power_pkg::*;
        import uvm_lp_core_pd_pkg::*;
        uvm_power_multicore pd;

        initial begin
                pd = new();
                pd.core_power_down();
        end
endmodule
```

Referring to the ARM Cortex A53 architecture different power domain such as PDCORTEXA53, PDL2, PDCPU, PDL1,etc. are consider and their relevant power routines functions are being called through UVM.

B. UVM Low Power Scenario Package

```
//THIRD SOURCE CODE

`include "uvm_lp_core_pd_pkg_dpi_c.sv"

package uvm_power_pkg;

class uvm_power;
      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;

      typedef enum {off,normal,standby,sleep,retention,dormant,
            deepsleep,ready,c0,c1,c2,c3,c4,c6,c7,c8}power_state;

  power_state state;
```

```
virtual function int powerup(state);
  begin
    case(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 sequential_power_down_up_multi_core_f();
endfunction

virtual function int power_up_another_core_f();
endfunction
```

UPVM - Implementation

UPVM Class for standard Low Power validation which is UVM Scoreboard which may be extended

Linux Shell/Python script

- To grep DUT directories for “modules” in System Verilog code, starting with “top”

System Verilog with link list class

- Code to read top
- Grep instantiated modules
- Build a tree of double link list with multiple head and tails

System Verilog Code

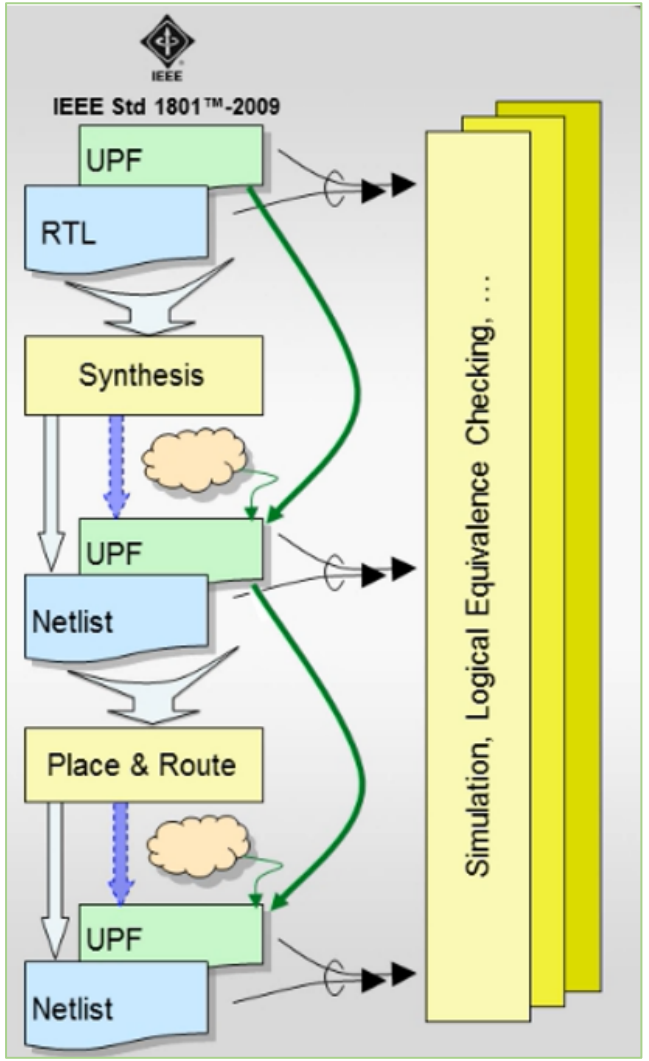
- To select “**set_design_top**” as well as Domains

Use Python scripts

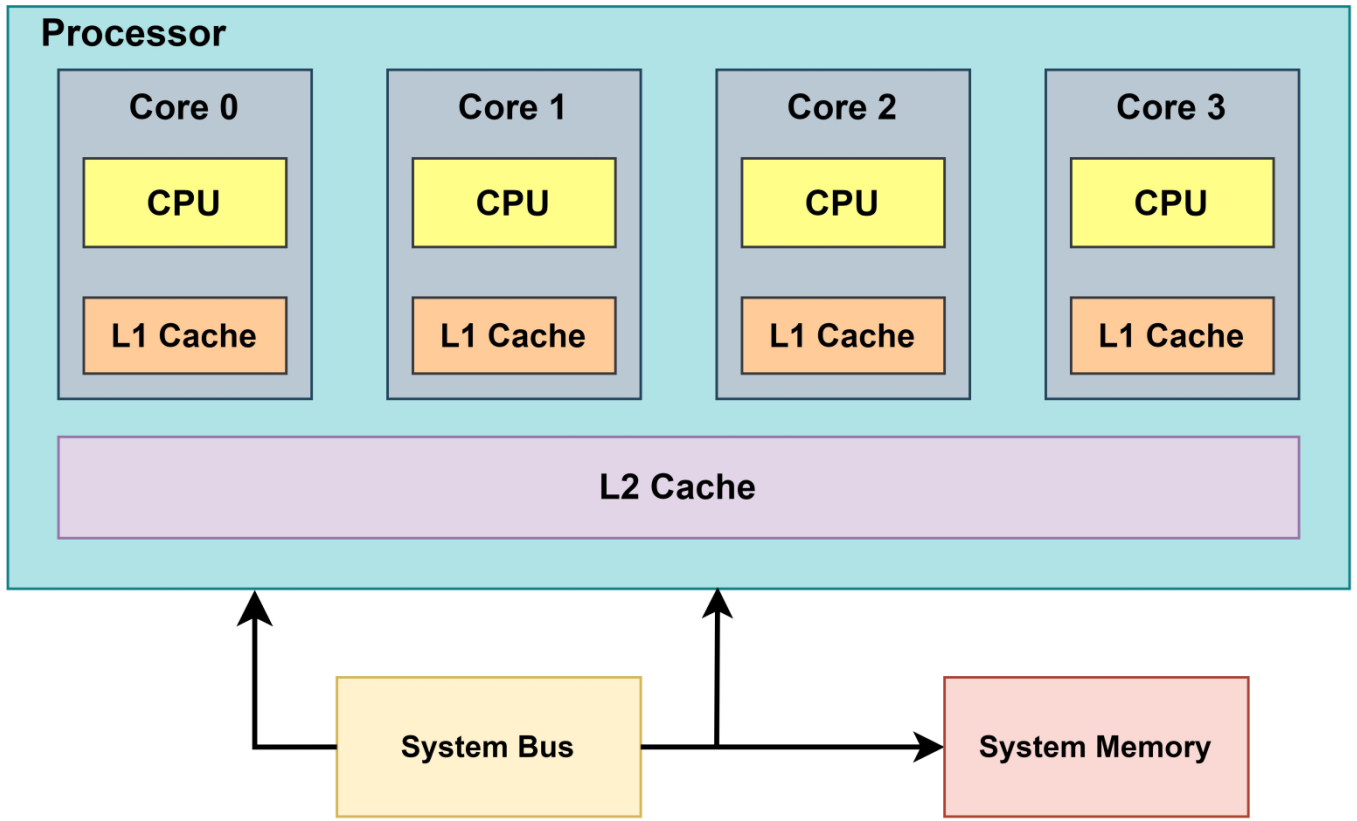
- Designer to identify Power Domains from a list of Module with sub elements
- Apply **“set_scope”** is to indicates the existing hierarchy.

UPVM to RTL, GLS, GDSII

- UPVM Class for standard Low Power validation which is UVM Scoreboard which may be extended
- Intent of UPVM is Low Power Verification – responsibilities end once we have verified the Design.
- Handover to Tier1 Industry leading tools for Low Power.
    - Python script to generate UPF file based on “Power Verification” in UPVM
    - Standard Low Power/Aware tools further the implementation flow from RTL, GLS and GDSII



Transparency for ARM multi-Core L1 & L2 Power Up and Power Down sequences for low power operations



Implementation Details for multi-Core Low Power Operations with L1 and L2 cache

- Incorporating ARM low-power principles:
  - Multi-cores already incorporate UPF concepts and provide API calls for managing multi-core operations and state transitions.

- Power Domain Classes (PDCORTEX, PDCPU, PDL1, PDL2, etc.) are available for the ARM Cortex A53 processor (as an example), facilitating power management in a multi-core environment.
- ARM Cortex processors have L1 and L2 cache memory levels

cache being small, fast, and low latency

secondary buffer for frequently accessed data

- **Power consumption is a concern for processor designers**, and ARM Cortex A53 allows power savings by turning off cores within a cluster.

Integration of L1 and L2 Cache using Package

- Scenario:
  - When a core is turned off-
    - Its cache data becomes inaccessible (the data needs to be transferred to L2 or system memory so that on core restart the data is restored)
    - A common enable can simultaneously power off the L1 and L2 cache to reduce power consumption. (both L1 & L2 data needs to be saved in system memory.

- The Power Management structure efficiently handles PowerDown and PowerUp operations by incorporating Power Domain Classes in compliance with ARM's recommendations.

- To sequence PowerUp/PowerDown for multi-Core L1 & L2 Cache, built-in ASM (Assembly Language) routines are utilized.

Integrated approach provides designers and verification engineers with a comprehensive strategy from the beginning of the design/verification process.

Controlling Signals for L1 and L2 Cache

L1 and L2 CACHE POWER CONTROL SIGNALS			
1	RESET Enable/Disable	DBGL1RSTDISABLE	L1 Reset Disable State=0 (Initial) to enable L1 Reset in PD
		L2RSTDISABLE	L2 Reset Disable State=0 (Initial) to enable L2 Reset in PD
2	Disable Data Cache	SCTLR.C (C- cache enable)	System Control Register Cache line bit C = 0
		HSCTLR.C (cache enable)	Hyper System Control register bit C = 0
3	Clean and invalidate cache	DCCISW.LEVEL =3'b000(L1)	DCCISW (Data cache clean and Invalidate by set way)
		DCCISW.LEVEL =3'b111(L2)	DCCISW.LEVEL = 3'b000 (Clean and Invalidate L1 Cache) and DCCISW.LEVEL = 3'b111 Clean and Invalidate L2 Cache
	Clean and invalidate cache by Virtual Address	DCCIMVACS	
	Memory Model Feature Register (MMFRI)	L1HvdVA, L1UniVA,L1HvdSW,L1UniSW,L1Hvd, L1Uni, L1TstCln,BPred; DCCISW_s set_way; DCCIMVAC_s virtual_addr:Bored	level 1 harvard cache by virtual address,level 1 unified cache by virtual address, level 1 harvard cache by set/way, level 1 unified cache by set/way, level 1 harvard cache, level 1 unified cache, level 1 cache test clean, Branch Predictor..
4	Disable Data Coherency	CPUECTLR.SMPEN	Low power retention state(CPU RETENTION CONTROL REGISTER .Switch Mode Power suplly Enable = 0 (Disable Data coherency))
5	L2 Cache Standby state	STANDBYWFIL2	the following conditions are met:
6	ACE READ LOCK L1 Mem	ARLOCKM	Read no snoop control signal ARLOCKM=1(For ACE Interface)(Inner/outer shareable Cache) and FOR Load/store
7	ACE WRITE LOCK L1 Mem	AWLOCKM	Write No snoop Control signal AWLOCKM= 1(For ACE Bus Interface transactions)(Inner/Outer write through) For load/store
8	Load No snoop	ReadNoSnp	Read no snoop control signal with Excl set High)(For CHI Bus Transaction Interface)
9	Store No snoop	WriteNoSnp	Write No snoop Control signal with Excl set high)(For CHI Bus Transaction Interafce)
10	Non snoopable	Non-snoopable	For non shareable cache operations
11	Bus Interface Configuration signals		Shareable, Non Shareable(Inner and Outer) Power domain control signals with / without L3 Memory
12	maintenance opertions	BROADCASTCACHEMAINT	When you set the BROADCASTINNER pin to 1 the inner shareability domain extends beyond the Cortex-A53 processor and Inner Shareable snoop and maintenance operations are broadcast externally. When you set the BROADCASTINNER pin to 0 the inner shareability domain does not extend beyond the Cortex-A53 processor. When you set the BROADCASTOUTER pin to 1 the outer shareability domain extends beyond the Cortex-A53 processor and outer shareable snoop and maintenance operations are broadcast externally. When you set the BROADCASTOUTER pin to 0 the outer shareability domain does not extend beyond the Cortex-A53 processor. When you set the BROADCASTCACHEMAINT pin to 1 this indicates to the Cortex-A53 processor that there are external downstream caches and maintenance operations are broadcast externally. When you set the BROADCASTCACHEMAINT pin to 0 there are no downstream caches external to the Cortex-A53 processor.
		BROADCASTOUTER	
		BROADCASTINNER	

UVM Low Power Package for L1 & L2 cache

```
class core_status_for_L1 extends uvm_power;
      L1_data_cache L1_dc;
      power_state p_states;

      virtual task core_status_t;
            if(p_states = off)
                  $display("turning off L1 data cache");
            else
                  $display("looks for next transaction for the core");
      endclass

      virtual task check_L1data_status;
            if(L1_dc != 0)
                  $display("data transferring to L1 to L2 cache");
            else
                  $display("called the core routine");
      endtask
endclass
```

```
class uvm_power_multicore_L2_cache extends uvm_power_core;
      //ARM power domain L2 signals
      struct PDL2_s {
            rand bit ON;
            rand bit RET;
            rand bit OFF;
            rand bit nL2RESET;
            rand bit rL2FLUSHREQ;
            rand bit L2FLUSHREQ;
            rand bit L2FLUSHDONE;
            rand bit L2RSTDISABLE;
      }
}
```

```
class uvm_power_L2_RAM extend uvm_power;
      //L2 Cache Standby state
      rand bit STANDBYWFIL2;
      //Read no snoop control signal ARLOCKM=1(For ACE Interface)
      (Inner/outer shareable Cache) and FOR Load/store
      rand bit ARLOCKM;
      //Write No snoop Control signal AWLOCKM= 1(For ACE Bus Interface transactions)
      (Inner/Outer write through) for load/store
      rand bit AWLOCKM;

      virtual task L2_cache_operation;
            if(STANDBYWFIL2 = 1'b1)
                  $display("asserted to indicate that the L2 memory system is idle");
                  $display("L2 will be able to access the data from other resources");
            if(L2_received_data = L1_send_data)
                  $display("checking the data matching status between L1 and L2");
      endtask
endclass
```

Summary

- 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 respin's 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.

- Proposed in-built Power Domain Classes as extension to UVM Package as Library into UVM Power, Memories, Bus Interface cores using Power Management Architecture & UPF Constraints may be implement to include Power Domain in UVM.

Conclusion

- The Environment for designing Low Power routines, applicable to multi-Cores, L1 & L2 cache addressing the growing demand for smaller and low power designs.

- Integration of Power Architecture: Emphasizing the importance of incorporating power architecture strategy and verification early in the design process to prevent costly re-spins and ensure adherence to time-to-market guidelines.

- UVM-based low power classes: Recommending the availability of low power classes for multi-Core in the UVM Libraries' low power extension, enabling SOC designs to benefit from a UVM-like verification test bench.

THANK YOU

Any Questions

Contact Emails:
avnita@siliconinterfaces.com
priyanka@siliconinterfaces.com
sastry@siliconinterfaces.com
heena@siliconinterfaces.com