2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Unified Architecture of L1 L2 Cache with Low Power Extensions for MultiCore UVM-based Library Package

Avnita Pal, Priyanka Gharat, Puranapanda Sastry, Darshan Sarode

Silicon Interfaces®
a software and vlsi design center

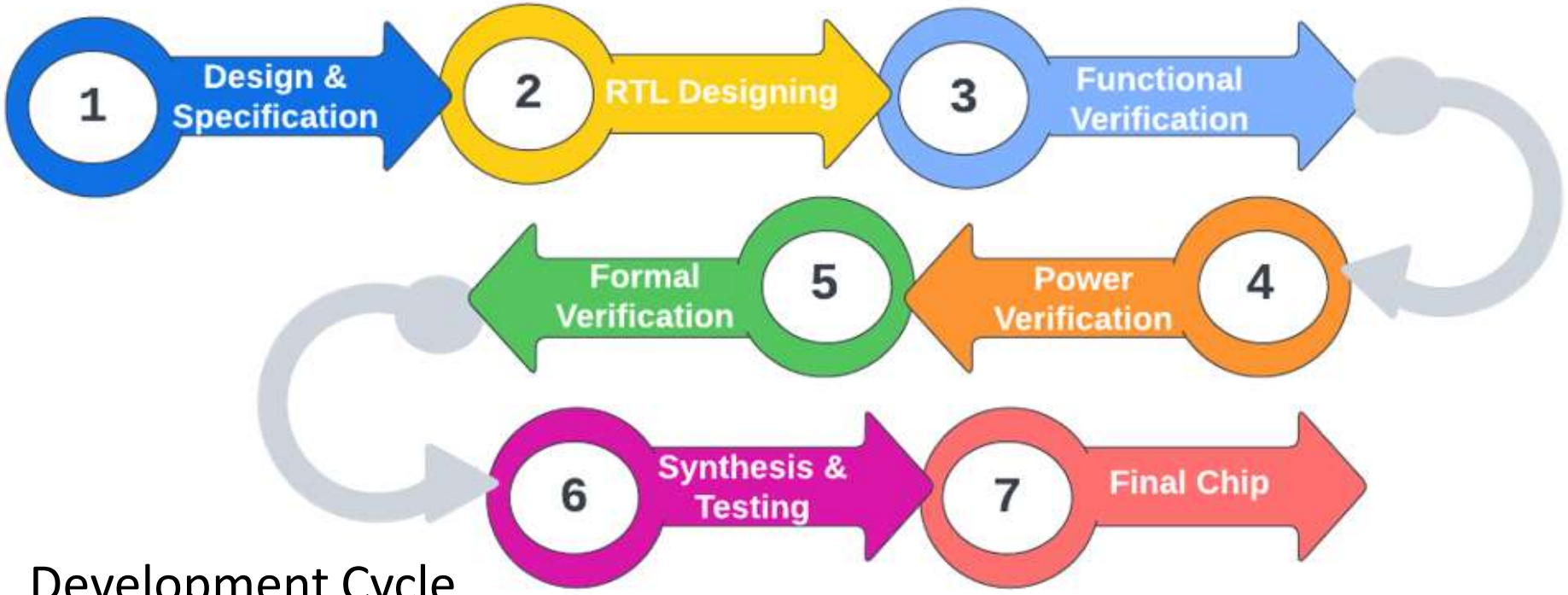
accellera SYSTEMS INITIATIVE

Agenda

- Problem Statement
- Solution
- The Challenges
- UVM Low Power Package Architecture
- Execution of Low Power utilization in multi-Cores
- Implementation and Integrated Flow
- Conclusion and Summary

Problem Statement

Can we optimize the low power verification process by integrating it into the earlier stages of power functional without compromising on the verification quality?

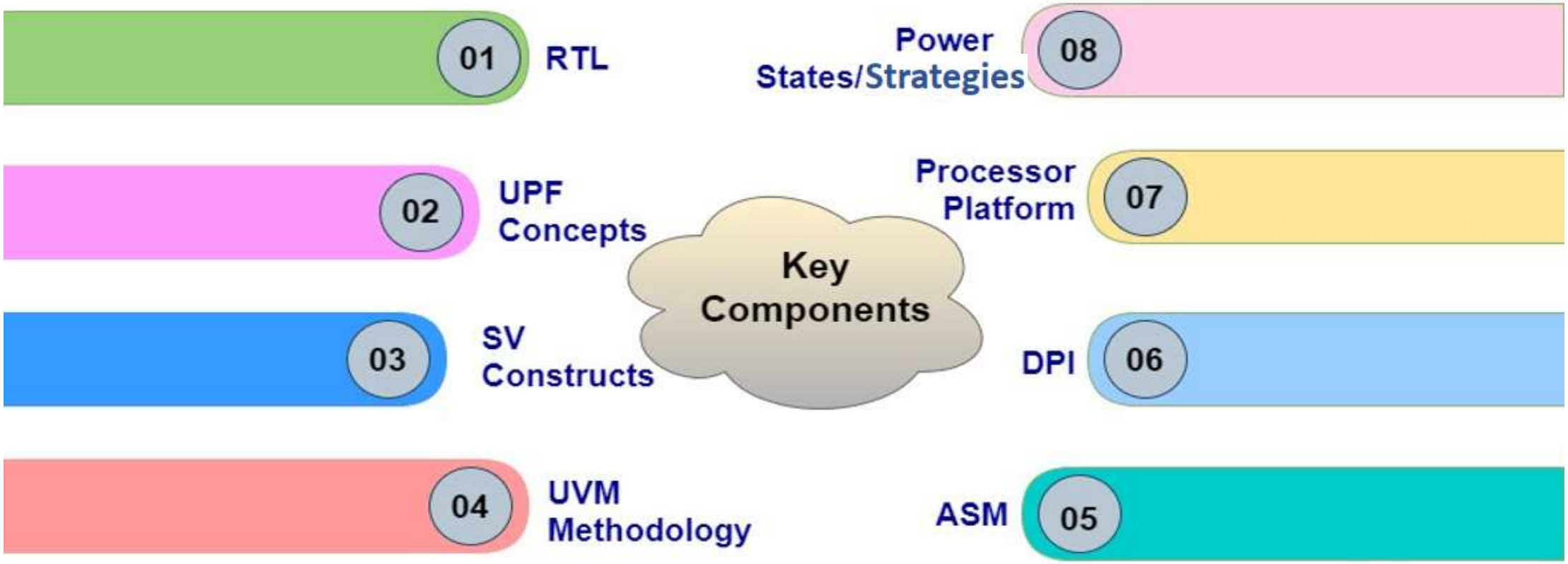


Development Cycle

Solution

- The current approach of incorporating Power Architecture after Functional Verification is not optimal. It should be integrated into the strategy from the beginning, along with Methodologies based Functional Verification and Coverage, and Low Power Implementation.
- The potential of a unified platform like UVM has empowered to create library components encompassing low-power strategies, functional methodologies, and UPF-based low-power architecture.
- Expanding the use of UVM-based Object Classes to include UVM-based Classes for Cores, Multi-Cores, Bus Interface for signals Memory and Devices.

The Key Components of the necessary approach



The Challenges

- The challenges associated with Power Verification

Integrating Design within Verification Methodologies

- Low Power is a Design activity
- EDA tools Validate the design

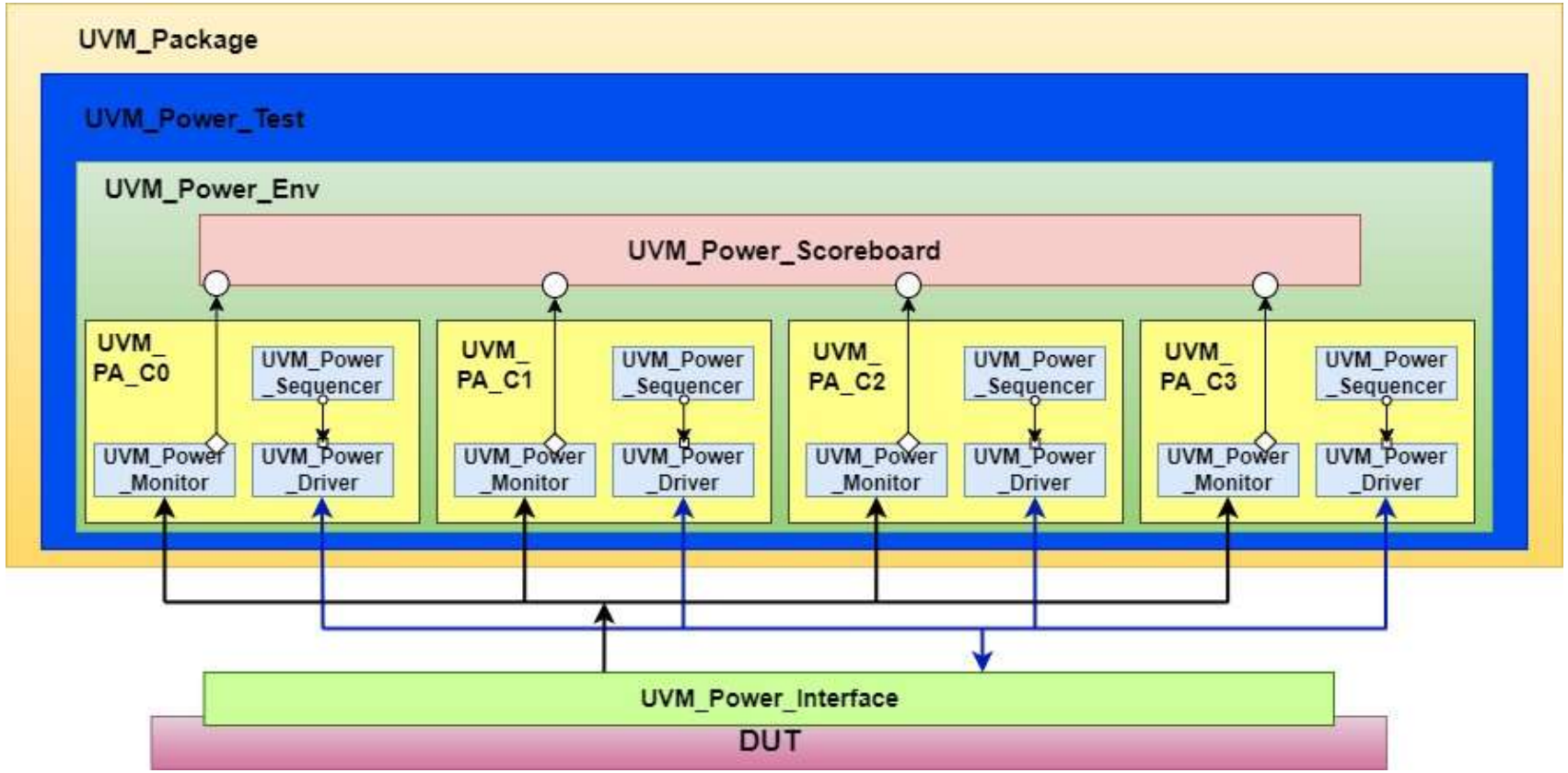
Power Validation is …

- At all stages of process – RTL, GLS, GDSII

Multi-Cores have …

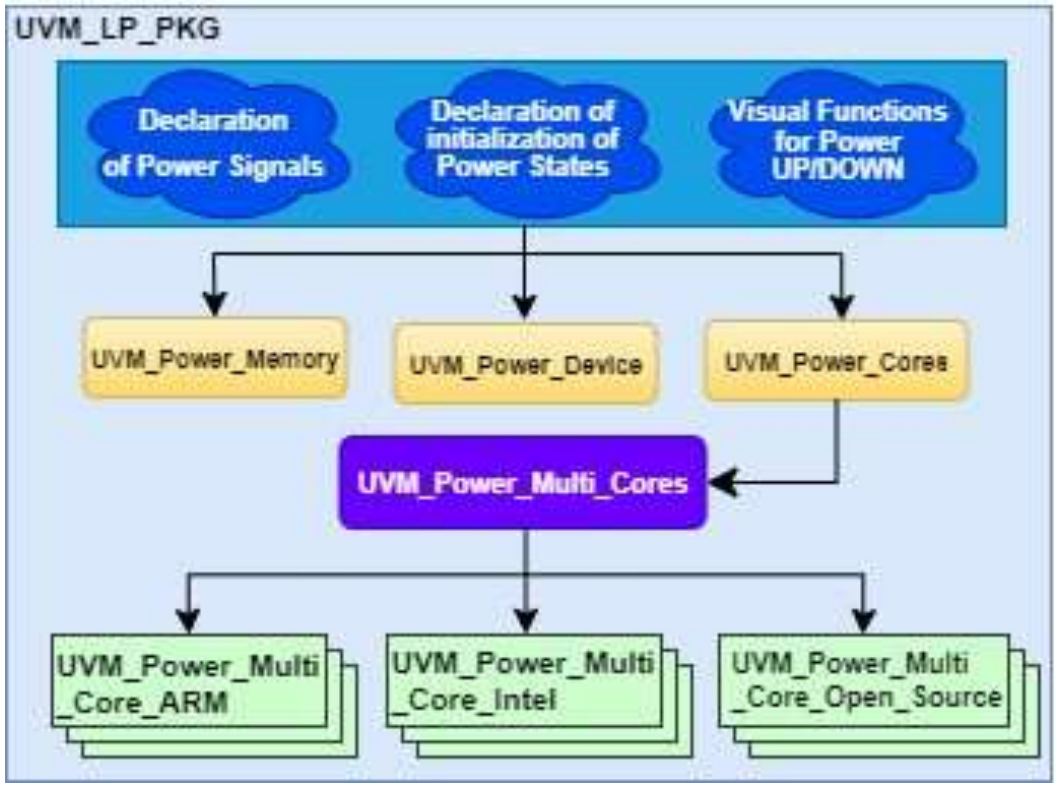
- Power designs integrated within the Design and accessible with Verification Bench

Recommended Use Model
Using multi-Core inbuilt Power State

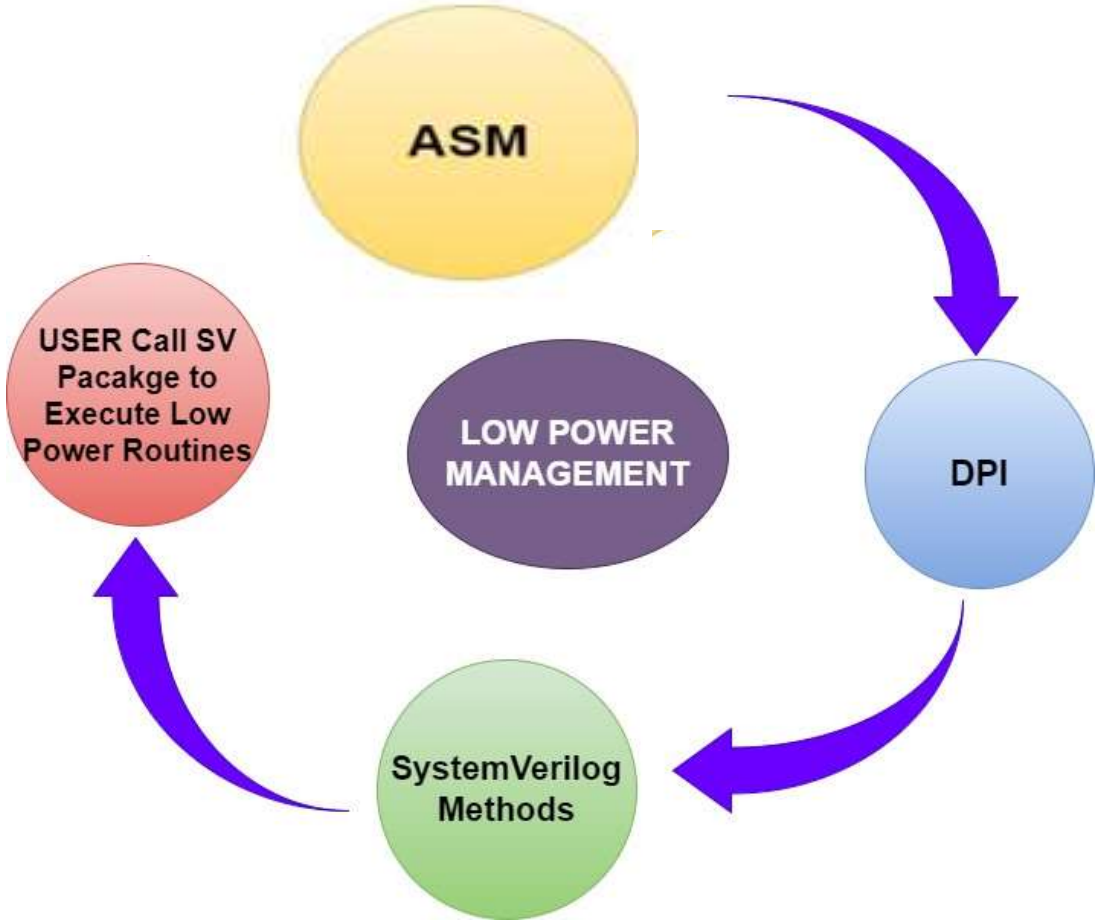


UVM Low Power Package Architecture

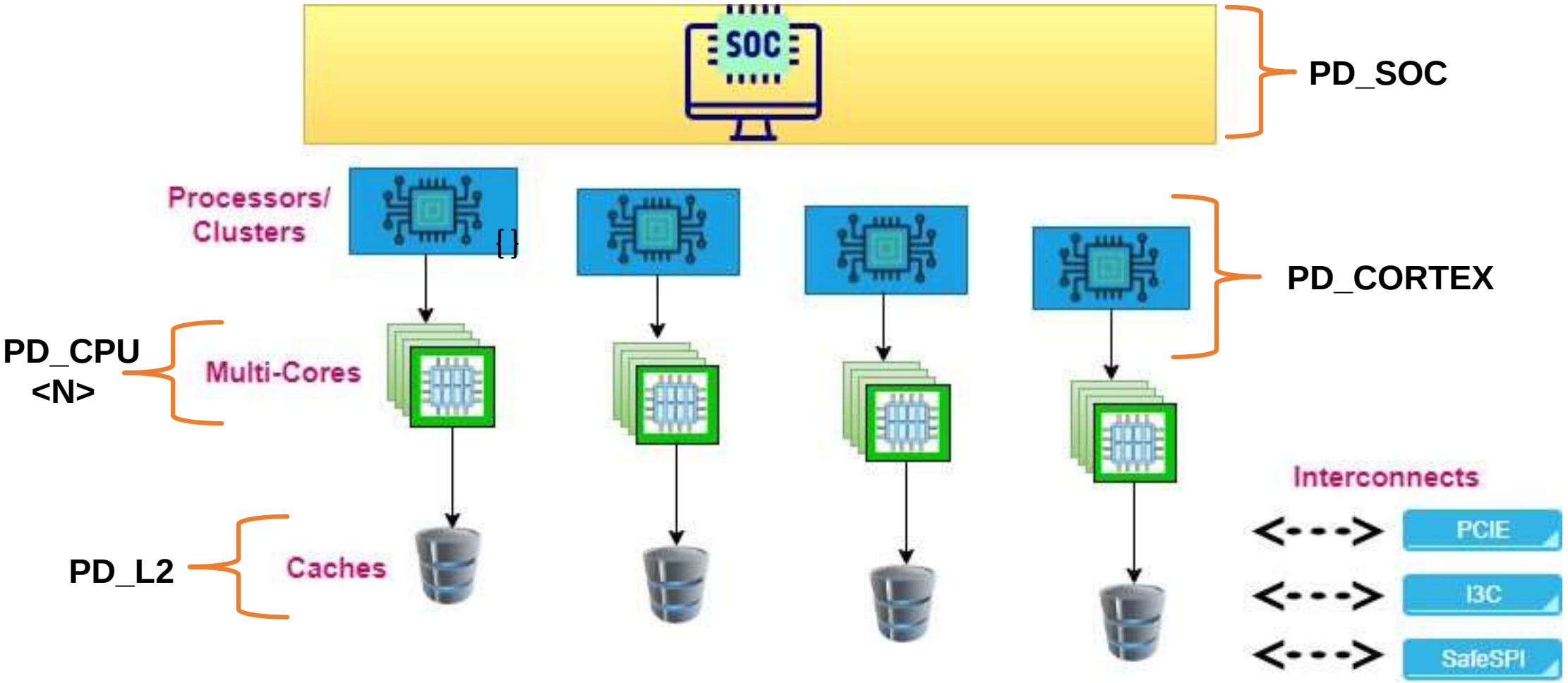
- An effective Power Management structure can be built using UVM classes that create power domains, define scopes, and provide nodes for each domain.
- These classes can be used as a library and expanded to match the device's architecture. By using these classes, the Power Management structure can be easily managed and customized to meet the device's requirements.
- The Power Management structure relies on the Power state of each domain, activating different virtual methods and sub-routines.



Execution of Low Power Utilization in Multi-Cores



Functional Description of Power Domains



Implementation of multi Core

- Multi-Core incorporates low-power principles, including UPF concepts, and provides API calls for managing multi-core operations and transitioning between states.
- Multi-Core has created Power Domains such as PD_CPU, PD_L1, PD_L2, and PD_CORTEX, which can be managed using register bit enabling and output clamps activation/deactivation, eliminating the need for UPF-based Power Domains.
- Power Domain Classes are extended to the Low Power UVM Package as a library, enabling their use in the Power Management architecture for Memories, Bus Interface cores, and other components.
- Multi-Core provides assembly code instructions for executing PowerUp and PowerDown routines, supporting multiple core states such as Ready(D3_Hot), Normal, Standby, Retention, Dormant, Deep Sleep (D3_Cold).

Integration of L1 and L2 Cache using Package

- The power management system efficiently handles turning off and on by using Power Domain categories in the setup.
- For the L2 cache, there's a special control bit called STANDBYWFIL2. This bit is crucial because it tells us if both the individual cores and the L2 memory system are in a power-saving idle state.
- When the STANDBYWFIL2 bit is set, it tells the power management system that everything, including the cores and L2 memory, is in an energy-efficient idle state. This is the right time to start saving power by shutting down the processor.
- To make this happen, we use DPI import statements in UVM classes, which indirectly call functions defined by multi-core. We'll explain these DPI calls in more detail later in the presentation.

Instructions 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:Bpred	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 ASM Routines

```
//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 {DMB,DSB,ISB}barrier_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:"); \
```

UVM Low Power DPI Package

```
//SECOND SOURCE CODE
`include "arm_cortexa53_assembly_code.c"
package uvm_lp_core_pd_pkg;

      typedef enum {CLEAN_BY_SETWAY,
            INVALIDATE_BY_SETWAY,
            CLEAN_INVALIDATE_BY_SETWAY,
            CLEAN_BY_VA_TO_POC,
            CLEAN_BY_VA_TO_POU,
            CLEAN_BY_VA_TO_POP,
            INVALIDATE_BY_VA_TO_POC,
            CLEAN_INVALIDATE_BY_VA_TO_POC,
            CACHE_ZERO_BY_VA,
            INVALIDATE_ALL_TO_POUIS,
            INVALIDATE_ALL_TO_POU,
            INVALIDATE_BY_VA_TO_POU
      }uvm_lp_core_cmo_type_e;

      typedef enum {DMB,
             DSB,
             ISB
      }uvm_lp_core_barrier_type_e;

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

```
import "DPI-C" function void core_status_t();
import "DPI-C" function void check_L1data_status();
import "DPI-C" function void L2_cache_operation();
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);
```

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
```

Results

- The result of the research is that the focus of the implementation is mainly on a multi-core low power environment. The full implementation needs to be done in a multi-Core environment in close collaboration and cooperation with multi-Core development environment. So, that would permit us to power up and power down cores. As we have made use of some system tasks to verify the operation or execution of the task with in extension of sv classes.
- Here, the role of ASM is essential for machine code insgtrauctions to test the low power environment. Integrating low power strategies within methods of UVM classes would help in verifying the low power aspects along with functional verification process concept like functional coverage, assertions as it gives additional advantage in the terms of verification of design.

Conclusion and Summary

- In the conclusion, we proposes the use of multi-Core ASM environment for designing Low Power routines for multi-Core as a case study, which can also be applied to other multi-Cores like Intel or ARC.
- The routines can be built for Bus Interface signals, Memory, and Device needs to be written, as the need for smaller and low power designs increase.
- The approach emphasizes the importance of implementing power architecture strategy and verification as an integral part of the design process, rather than an afterthought post-functional verification, to avoid unwanted re-spins that can be detrimental to costs and time-to-market guidelines.
- The presentation concludes by recommending that low power classes for multi-Core should be available in the low power extension of UVM Libraries to enable SOC designs to have a UVM-like verification test bench.

Thank You!
Questions?