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

Avnita Pal, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (*avnita@siliconinterfaces.com*)

Priyanka Gharat, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (*priyanka@siliconinterfaces.com*)

Sastry Puranapanda, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (*sastry@siliconinterfaces.com*)

Abstract— **This Paper demonstrates the continuum for multi-Core architecture integrating UPF based Low Power methodologies and strategies to L1 & L2 Cache in Off, Sleep, Dormant and Retention modes within the UVM Low Power Package by addressing limitation of previous works (referenced) to incorporate multi-Core low power libraries (which has the classes for SOC environment Devices, Buses and Memory) for low power strategies which may be deployed in UVM Agents executing within Run Phase with in-built ASM routines to sequence PowerUp/Down for multi-Core L1 & L2 Cache and incorporate these within low power UVM classes using SystemVerilog and DPI.**

Keywords— Power Management, Low Power, UVM (Universal Verification Methodology), Functional Verification, SystemVerilog, Unified Power Format (UPF), L1 Cache, L2 Cache, DPI (Direct Programming Interface), PowerUp, PowerDown, Assembly Language (ASM), SoC (System on Chip)

I. INTRODUCTION

The current approach of incorporating Power Architecture after or in parallel to 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. In previous works, a single platform, such as UVM, was used to develop library components for devices, including low power strategies, Functional Verification Methodology, and UPF-based Low Power Architecture. This allowed designers and verification engineers to have a comprehensive strategy from the start of the design/verification process.

Why not expand the use of UVM-based Object Classes to include UVM-based Classes for Low Power for L1/L2 Cache along with for Cores, Multi-Cores (such as ARM, Intel, and Open Source), Bus Interface for signals (such as AMBA AXI, CHI, PCIe, and Wishbone), Memory and Devices? These could be registered for reusability and constructed within the UVM Environment. These classes could then be utilized in the Build, Connect, and Run phases, providing SOC Verification Engineers with ready-to-use classes that can be extended to the SOC Design being tested.

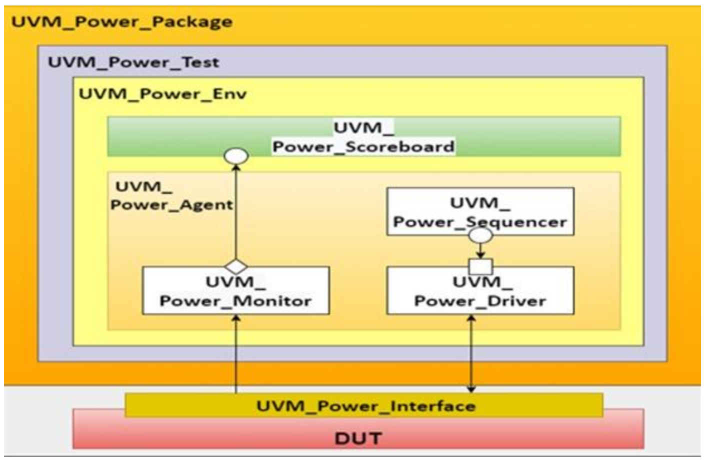


Figure 1. UVM Low Power Hierarchical Structure

II. Implementation Details And Flowchart

The creation of an effective Power Management structure can be achieved by utilizing UVM classes that include various tasks such as creating power domains, defining different scopes, and supplying nodes for each domain. These classes can be used as a library and extended to create structures based on the architecture of the device under test. By incorporating these classes, the Power Management structure can be efficiently managed and adapted to suit the specific needs of the device.

In this paper, we are extending the work as done and presented in earlier paper for multi-Core. The Power Management structure is dependent on the Power state of each domain, which triggers various virtual functions, tasks, and sub-routines. A top-level UVM_power_pkg is incorporated into the test bench for a multi-core and extended based on the specifications defined in UPF to perform PowerUp and PowerDown of any Core sequences. 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.

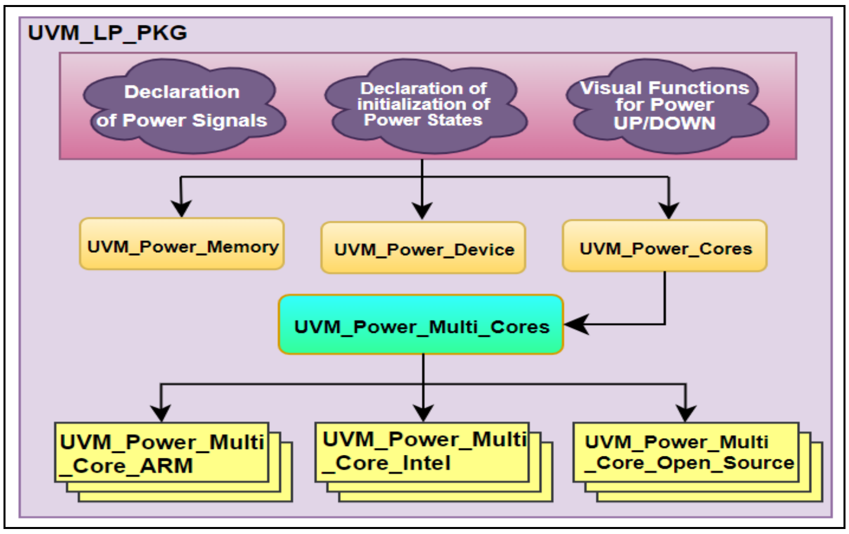

UVM_LP_PKG
Declaration of Power Signals
Declaration of initialization of Power States
Visual Functions for Power UP/DOWN
UVM_Power_Memory
UVM_Power_Device
UVM_Power_Cores
UVM_Power_Multi_Cores
UVM_Power_Multi_Core_ARM
UVM_Power_Multi_Core_Intel
UVM_Power_Multi_Core_Open_Source

Figure 2. Architecture of Low Power UVM Package Library

III. Implementation Details For Multi-Core Low Power Operations with L1 and L2 cache

ARM has incorporated low-power principles, including UPF, and provided API calls for managing multi-core operations and transitioning between states. The company has also 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, making it unnecessary to create UPF-based Power Domains. These features are leveraged to manage power-down and power-off operations in compliance with ARM's recommendations. 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. The Power Domain classes for the ARM Cortex A53 processor, such as PDCORTEX, PDCPU, PDL2, and so on, are created by ARM and can be utilized in the multi-core environment through factory registration in build_phase, run_phase, and connect_phase.

The Arm Cortex processors incorporate two levels of cache memory - L1 and L2. L1 cache is small and fast and has low latency providing quick access to frequently used data and instructions. L2 cache has high latency which then makes it slower but still faster than main memory, and acts as a secondary buffer for frequently accessed data.

Power consumption has been a major concern for processor designers as it is a key factor affecting battery life in portable devices. In the ARM Cortex A53 processor we can save power by turning off some cores in a cluster. When a core is turned off, its cache data becomes inaccessible, and it would take a long time to reload the data if required. To address this issue, we can use common enable to turn off the L1 and L2 caches simultaneously, as our core itself is shut down so the L1 cache is out of the picture and only L2 Cache will get accessed and can be power-off process.

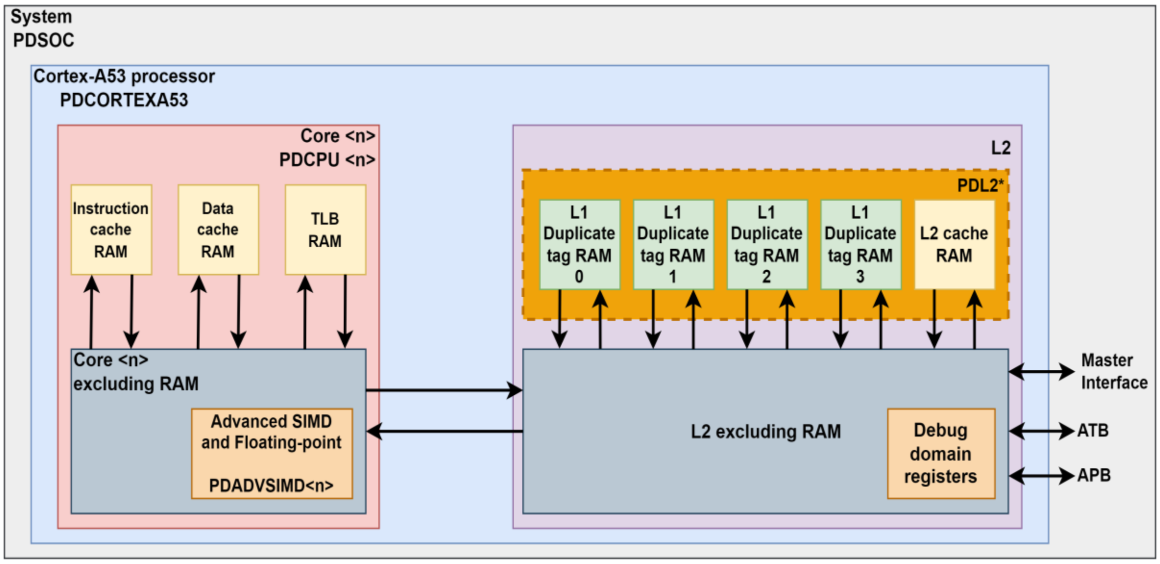
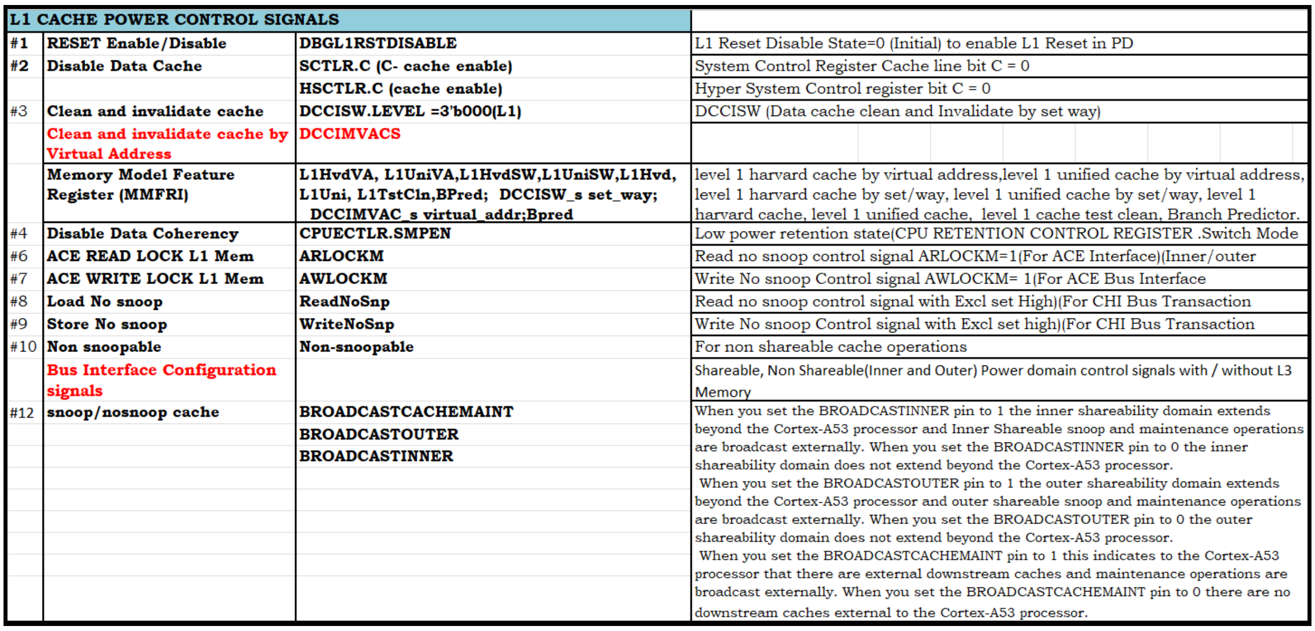


Figure 3. ARM Cortex A53 Power Domain Block Diagram

By incorporating these classes, the Power Management structure can efficiently manage power-down and power-off operations in compliance with ARM's recommendations. The use of these classes enables the creation of a comprehensive Power Management structure, with support for multiple cores and power domains. The architecture can be implemented using the Low Power UVM Package as a library, making it easier to integrate into the verification process.

To sequence PowerUp/Down for multi-Core L1 & L2 Cache there is need to utilize the in-built ASM (Assembly Language) routines. The package incorporates two levels of cache memory - L1 and L2. L1 cache is on board cache which is small and fast in transferring and fetching the data with low latency and L2 cache is the external cache its integrated with SCU (Snoop Control Unit) controlling up to 4 cores within the clusters to provides the duplicate copies L1 data cache tags for coherency support and high latency which acts as a secondary buffer for fast accessed data. The sequences for this is being shown in the spreadsheet below. This integrated approach empowers designers and verification engineers with a comprehensive strategy right from the initiation of the design/verification process.



L1 CACHE POWER CONTROL SIGNALS			
#1	**RESET Enable/Disable**	**DBGL1RSTDISABLE**	L1 Reset Disable State=0 (Initial) to enable L1 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)
	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
#6	**ACE READ LOCK L1 Mem**	**ARLOCKM**	Read no snoop control signal ARLOCKM=1(For ACE Interface)(Inner/outer
#7	**ACE WRITE LOCK L1 Mem**	**AWLOCKM**	Write No snoop Control signal AWLOCKM= 1(For ACE Bus Interface
#8	**Load No snoop**	**ReadNoSnp**	Read no snoop control signal with Excl set High)(For CHI Bus Transaction
#9	**Store No snoop**	**WriteNoSnp**	Write No snoop Control signal with Excl set high)(For CHI Bus Transaction
#10	**Non snoopable**	**Non-snoopable**	For non shareable cache operations
	Bus Interface Configuration signals		Shareable, Non Shareable(Inner and Outer) Power domain control signals with / without L3 Memory
#12	**snoop/nosnoop cache**	**BROADCASTCACHEMAINT** **BROADCASTOUTER** **BROADCASTINNER**	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.

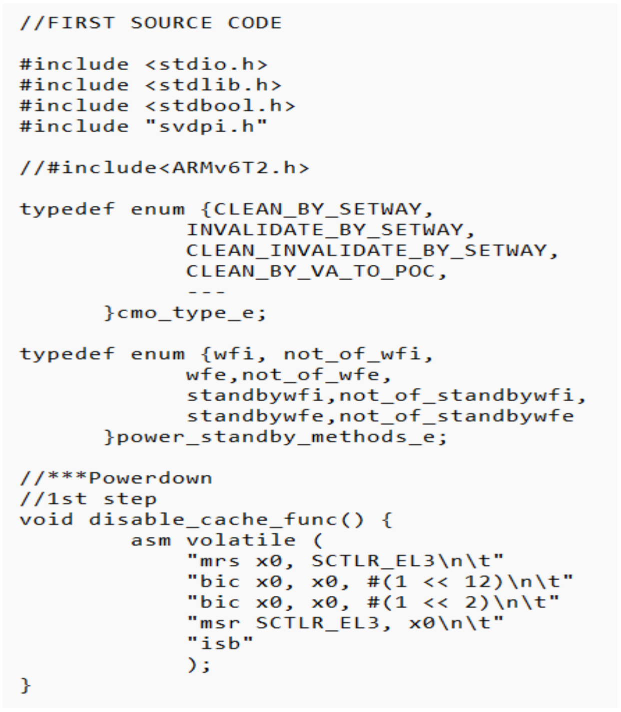
ARM provides assembly code instructions for executing PowerUp and PowerDown routines. The ARM Cortex A53 Processor/Cluster development environment supports multiple core states, including Ready(D3_Hot), Normal, Standby, Retention, Dormant, Deep Sleep (D3_Cold), which can involve one, two, or even three-step state transitions. Power Up and Power Down routines are called to operate on single or multiple cores and are implemented in both C Language and assembly language. The action on the L1 & L2 caches will be dependent on the current state of the Core and to which state it needs to go. For example, Normal to Standby or even from Retention to Off. The L1 may be PowerDown but the L2 may be still on and when the PowerDown sequence is initiated the memory needs to be retained in system memory.

To leverage and reuse the C code, these Power Down routines are imported into the low power UVM SV package using Direct Programming Interface (DPI-C). This interface allows for the calling of C functions from SystemVerilog using "DPI" declarations. By importing C code into the UVM SV package, it is possible to reuse existing code and take advantage of the functions implemented in C to improve the efficiency and effectiveness of the verification process.

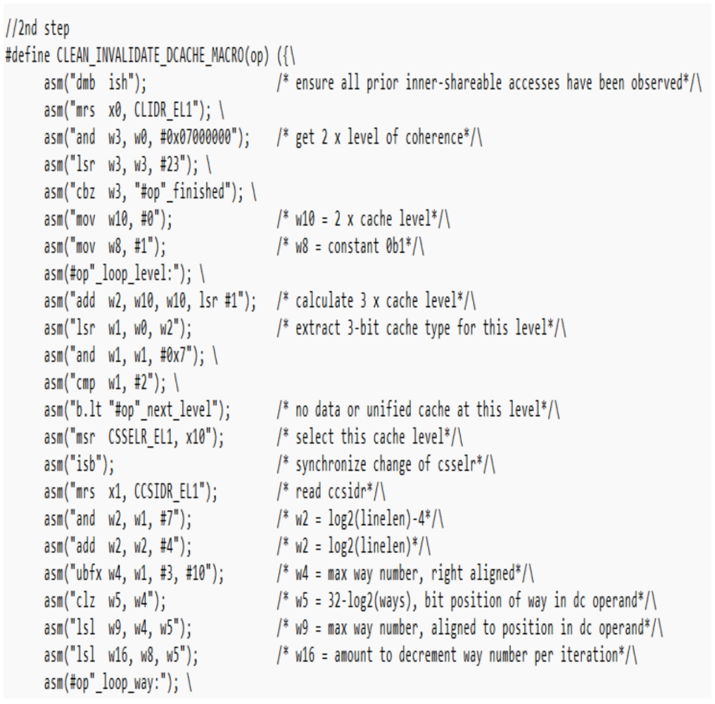
IV. PRE LIMINARY RESULTS AND SOURCE CODE

In sub-sections "A", "B", "C", and "D," we demonstrate our approach to utilizing ARM multi-Core L1 & L2 Power Up and Power Down sequences for low power operations. We achieve this by using ASM and wrapping it within SystemVerilog-based classes through DPI and C. Our routines utilize ARM Core Assembly language for PowerDown and PowerUp of the core or multi-Core, along with its Low Power Management functions. These ARM multi-Core functions are accessed using ASM within the ARM Development Environment, which includes the commented include statement for "ARMv6T2.h," that would be uncommented to run in the environment. We call ARM ASM code in C functions, which are then utilized in SystemVerilog through DPI for extending classes in Low Power extension to UVM.

In summary, by transferring the L1 cache data to the L2 cache before powering off a core, the ARM Cortex A53 processor can save power while preserving the data and maintaining the overall performance of the processor. This technique has been widely used in portable devices where battery life is a critical factor, such as smartphones, tablets, and other IoT devices.



```
//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:"); \
```

A. *UVM Low power DPI package*

This Assembly Code is being enveloped using DPI calls into anSV package by declaring source file for SystemVerilog package *"uvm_lp_core_pd_pkg"*. The C code shown in section A is being called inside "*uvm_lp_core_pd_pkg"* package using import *"DPI-C"* keyword. This helps to provide connectivity using ARM routines defined using C assembly language in System Verilog.

```
//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);
```

B. *UVM Low power Scenario Package*

The below *uvm_power_pkg* package includes SystemVerilog file which imports DPI functions mentioned in *uvm_lp_core_pd_pkg* package. Further the same class is being registered in UVM factory. It also includes the members and methods which performed all the power related routines.

```
//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;
```

```
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
```

```
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
```

```
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
```

C. Functional Description for Power Domains for PowerUp and Power Down

Referring to the architecture of ARM Cortex A53 using different power domains such as PDCORTEXA53, PDCPU, PDL1, PDL2, etc. are considered and their relevant power routines functions are being called through UVM as shown in below source code.

In the sample test case, the user can utilize the library packageat different levels. Class and Functions described in section A, B and C are being called in *uvm_power_multicore* class

```
import uvm_lp_core_pd_pkg::*;

class uvm_power_multicore extends uvm_power_core;
      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;

      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
```

```
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;
      }
}
```

This class shall be registered in UVM factory of low power package. The full implementation needs to be done in an ARM environment in close collaboration and cooperation from ARM in the ARM Cortex Development environment. So, that would permit us to PowerUp and PowerDown L1 & L2 Cache. In this paper, the outputs are being observed using $display and C printf (using DPI-C) to check the results. Further, the assembler code which is essential for testing will run on ARM Development Environment.

V. Conclusion

In conclusion, the paper proposed the use of ARM 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 paper suggests that routines can be built for Bus Interface signals, needs to be written, as the need for smaller and low power designs increase. The paper 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 paper 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.

VI. References

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

[2] Guide to change in IEEE1801-2013(UPF2.1)(techdesignforums.com)

[3] Arm Cortex-A53 MPCore Processor Technical Reference Manual r0p4

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

[5] Low Power Classes as extension to UVM Package Library by Shikhadevi Katheriya, Avnita Pal, et al, 59th Design Automation Conference, San Francisco, United States

[6] Leveraging UVM-based Low Power Package Library to SOC Designs
Shikhadevi Katheriya, Silicon Interfaces; Avnita Pal, Silicon Interfaces; Sastry Puranapanda, Silicon Interfacs