Leveraging UVM-based Low Power Package Library to SOC Designs

Shikhadevi Katheriya, Silicon Interfaces, Mumbai, Maharashtra, India (shikha@siliconinterfaces.com)
Avnita Pal, Silicon Interfaces, Mumbai, Maharashtra, India (avnita@siliconinterfaces.com)
Puranapanda Sastry, Silicon Interfaces, Mumbai, India (sastry@siliconinterfaces.com)
+91 2246008566
www.siliconinterfaces.com

***Abstract* - Consolidating Power Management architecture within UVM methodologies alleviates the divide for functional verification and the power management engineers. As using different languages, many times with different team members using different tools and divergent approaches there is scope of errors, discrepancy and ideas. Incorporation of Functional Verification and Power Management will lead to reduced verification time and better chance to meet the time to market deadlines, and therefore a Low Power UVM Package has already been developed as UVM Package as the Library and extended the same to UVM-based Low Power libraries. The aim of the paper is to use the proposed in-built Power Domain classes in System Verilog (UVM_Power) as Extension to UVM Low Power Package Library Bus Interface for signals (AMBA AXI, CHI, PCIe, and Wishbone), Memory and Devices. However, for Cores, multiCores, (ARM, Intel, Open Source, Etc.), the Low Power Designs are in-built to the developer environment and hence, the UVM structure proposed in this paper incorporates the C Library within SV Packages. These Power Libraries are offered as Base Class which may extend to extend the UVM Agent, UVM Driver and UVM Sequencer. The idea is to have Low Power objects available to the Designer of an SOC which may be inherited and further extended.**

I. INTRODUCTION

Incorporating Power Architecture seems to be more like an afterthought post Functional Verification, almost a fourth dimension to our strategy leveraging the test bench architecture. It would be more efficient to do Methodologies based Functional Verification and Coverage along with Low Power Implementation.

The library components with low power strategies, including UPF interleaves Functional Verification Methodology and Low Power Architecture in a single existing and widely deployed methodologies-based platform, like UVM, and 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. So, why not extend these UVM-based Object Classes to further incorporate UVM-based Classes for Cores, Multi-Cores, (ARM, Intel, Open Source, Etc.), Bus Interface for signals (AMBA AXI, CHI, PCIe, and Wishbone), Memory and Devices, which may be factory registered for reusability and then constructed within the UVM Agent, UVM Sequencer, UVM Driver, UVM Monitor and the UVM Scoreboard. Further these Class may be initiated in be used in Build, Connect and Run Phase. 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. Further, we have reviewed "UVM and UPF: an application of UPF Information Model" [5] in reference below. Both papers with reference [5] and [6] are proposing UPF construct within UVM and *this paper* is proposing Low Power extension for multiCores, for example ARM multiCore. This will permit the SOC Verification Engineer ready classes (for Devices and IPs using UPF construct UVM Classes and for multiCore UVM Low Power Classes for cores using SV interfacing to C using DPI) which may be extended to the SOC Design under Test.

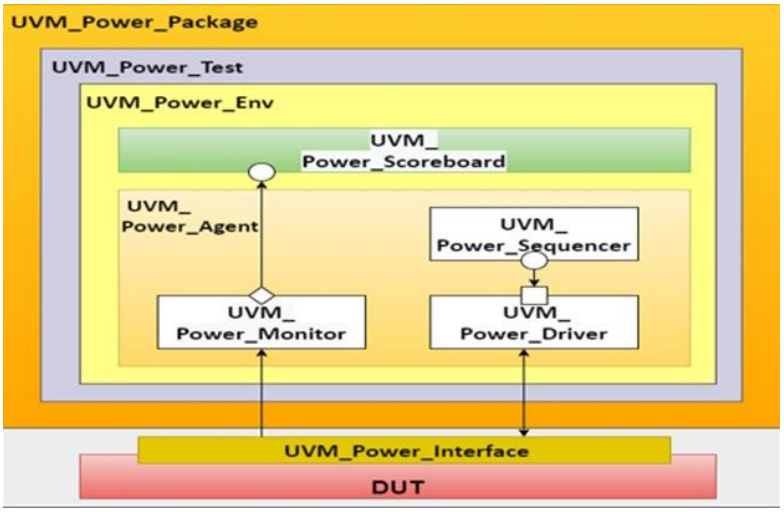

Figure 1. UVM Low Power Hierarchical Structure

II. Implementation Details/Flow Chart

An overall UPF structure is created using UVM classes which include different task as creating power domain, different scopes then supply nodes for each of the domain which are created. These classes are used as library and can be extended for creating structure based on DUT/SOC Architecture. Based on the Power state of different Power domain various virtual functions, task and sub-routine are being called. This top level UVM_power_pkg is incorporated in test bench for a multi-Core and further extended based on specification defined to do PowerUp and PowerDown of any Core sequences.

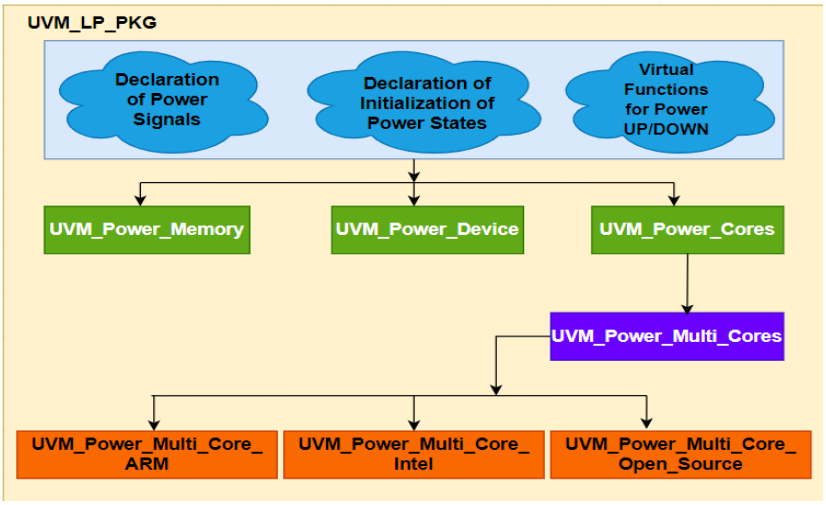

Figure 2 Architecture of Low Power UVM Package Library

III. Implementation Details For ARM Multicore Low Power Operations

ARM multi-core operations and how ARM handles low power and transition from states: ARM has incorporated principles of low power (perhaps including UPF) and given us API calls to transition the cores within the processors, as well as given us Power Domains, like PDCPU, PDL1, PDL2, PDCORTEX, Etc. So, to that extent it is not necessary to create UPF based Power Domain. Even, the Power Rails connectivity and ON/OFF are set by Registers

bit enabling and Output Clamps Activate/Deactivate. So, we are leveraging the process as asked by ARM for its Core Powerdown/up, etc.

Even in Multicore UVM Low Power Classes environment is followed by factory Registration, phases which includes build-phase, run_phase and connect phase.

ARM has Assembly Code instructions to be executed for any PowerUp and PowerDown routines. Further, an ARM Core may be in several states (Ready(D3_Hot), Normal, Standby, Retention, Dormant, Deep Sleep (D3_Cold), Off) which may be a 1-step or 2-step and even a 3-step state transitions. We are calling the Power UP and Power Down routines which all run in the ARM Cortex A53 Processor/Cluster development environments and these routines are forming the operation on single core or multiple cores. These routines are for power down the core which is written in C Language and ASM's are written in assembly language. This C Power down routines is then imported in low power UVM SV package through DPI-C which is Direct Programming Interface. It allows us to leverage and reuse existing C code. Functions implemented in C can be called from SystemVerilog using import "DPI" declarations.

IV. Preliminary Results

Here in the following source code for 1. C routines for power DOWN/UP source file are "arm_cortex _a53_assembly_code", which contains all the C functions with ASM code for cortexA53 processor/cluster with 64-bit ISA. This Assembly Code is being enveloped using DPI calls into a SV Package by declaring source file for SystemVerilog Package "uvm_lp_core_pd_pkg", which contains SV functions which is calling the C functions with the help of DPI import.

Extending UVM_power package to multicore

*A. C routines for Power down(**arm_cortex_a53_assembly_code.c**)*

These routines are for ARM Core Assembly language for PowerDown and PowerUp routines are written in C language. These functions written in C language are being incorporated in SV Power package. The implementation of the same is being shown in Section B. *UVM Low power DPI package*. Further these C functions are being called from SV, well actually from UVM, which will be integrated into the UVM Libraries.

```
#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,
                .....
        }cmo_type_e;
```

```
//----------Routines for POWER DOWN----------

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"
            );
}
```

```
#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*/\
      ........
      })
```

```
void clean_invalidate_dcache_func(cmo_type_e cmo_type){
        switch (cmo_type) {
                case CLEAN_BY_SETWAY:
                        clean_dcache();
                        break;
                .......
                default:
                        break;
        }}

void cpu_extended_contrl_reg_func() {
        asm volatile (
            "mrs x5, ACTLR_EL2\n\t"
            "bic x5, x5, #(0 << 1)\n\t"
            "mrs x6, ACTLR_EL3\n\t"
            "bic x6, x6, #(0 << 1)\n\t"
             .......
        );}

void barrier_func(barrier_type_e barrier){
        switch (barrier) {
                case DMB:
                        barrier_dmb();
                        break;
                .......
        }}
```

```
void WFI_hint_instruction_func(){
        asm volatile (
                "mrs x0, SCTLR_EL3\n\t"
                "bic x0, x0, #(1 << 11)\n\t"
                "bic x0, x0, #(1 << 10)\n\t"
                "bic x0, x0, #(1 << 9)\n\t"
                "bic x0, x0, #(1 << 8)\n\t"
                 .......
        );}

void WFE_hint_instruction_func(){
        asm volatile (
                "mrs x0, SCTLR_EL3\n\t"
                "bic x0, x0, #(1 << 11)\n\t"
                "bic x0, x0, #(1 << 10)\n\t"
                "bic x0, x0, #(1 << 9)\n\t"
                "bic x0, x0, #(1 << 8)\n\t"
                 ......
        );}

void transition_func(power_standby_methods_e wf){
        switch (wf) {
                case wfi:
                        WFI_hint_instruction_func();
                .......
                default:
                        break;
         }}
```

```
//----------Routines for POWER UP----------
void cpu_processor_rst_debug_access_func(bool DBGPWRDUP,bool nCPUPORESET){}
void power_pdcpu_func(bool PDCPU,bool DBGPWRDUP,bool nCPUPORESET){}
void clamp_release_func(bool CLAMPCOREOUT){}
```

B. 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
```

C. UVM Low power package

In Below uvm_power_pkg package we have the included the SV file which call to DPI functions mentioned in uvm_lp_core_pd_pkg package. Further this class is being registered in UVM factory. It also includes member and methods are also defined to perform all the power related routines.

Within the UVM Low Power Package, we are now defining Classes with virtual Functions and Tasks and these Classes may be extended by UVM Develops for SOC, incorporating ARM multiCores into the existing UVM Test Bench and extending and reusing the virtual functions

```
`include "uvm_lp_core_pd_pkg_dpi_c.sv"

package uvm_power_pkg;

class uvm_power;

        //This is the factory registration for low power uvm
        `uvm_lp_object_utils(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 {off,normal,standby,sleep,retention,dormant,deepsleep,
                        ready,c0,c1,c2,c3,c4,c6,c7,c8}power_state;

        power_state state;

        virtual function sequential_power_down_up_multi_core_f();
        endfunction

        virtual function int power_up_another_core_f();
        endfunction
endclass
```

D. *Functional Description for Power Domains for Power UP and Power DOWN*

Referring to the below 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 as shown in below source code. This is sample code taken further

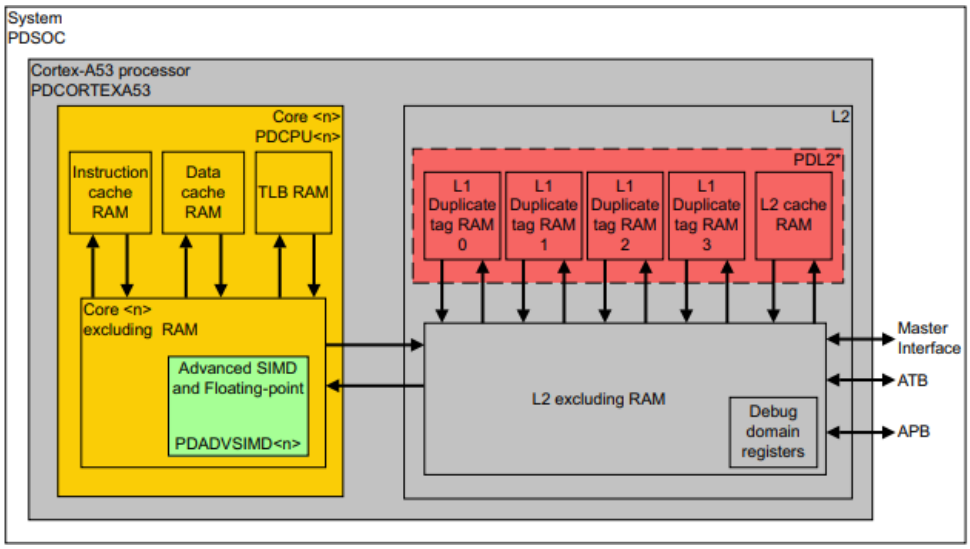

Figure 3. ARM Cortex A53 Power Domain Block Diagram

```
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
```

In the below Sample Test, the User Level utilization of the library package, Class and Functions described in section A, B, C and D are being called in uvm_power_multicore class. This class is also registered in UVM factory of low power package. In the task power down routines are called Low power package.

```
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
```

V. RESULT

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 actually PowerUp and PowerDown cores. In this paper, we have observed the outputs using $display and C printf (using DPI) check the results. We are not showing those statements since they are only for validation purposes. Further, In the Assembler code which is essential for testing will run on ARM Development Environment

VI. CONCLUSION

Proposed in-built Power Domain Classes as extension to UVM Low Power Package as Library for Cores, multiCores, (ARM, Intel, Open-Source Cores, Etc.), using Power management architecture shall bring in Power Verification at an earlier stage will bring down the total time for incorporating power strategies resulting in far shorter design cycles for SOC Designs. The Paper is considering ARM multi-core as a case study, but the same concepts may be applied to Intel, ARC or any Open-Source Cores). Further, routines for Bus Interface for signals (AMBA AXI, CHI, PCIe, and Wishbone), Memory and Device needs to be written (like UPF type Power Domain architecture as previously proposed in [5] and [6] as given in Reference). 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/time to market guidelines.

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] 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] UVM and UPF: an application of UPF Information Model by Amit Srivastava, Harsh Chilwal, Srivatsa Vasudevan in DVCON 2019
[6] Low Power Classes as extension to UVM Package Library by Shikhadevi Katheriya, Avnita Pal, et al, 59th Design Automation Conference, San Francisco, United States