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

Avnita Pal, Priyanka Gharat, Puranapanda Sastry, Darshan Sarode
Agenda

1. Problem Statement
2. Solution
3. The Challenges
4. UVM Low Power Package Architecture
5. Execution of Low Power utilization in multi-Cores
6. Implementation and Integrated Flow
7. 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?
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

- PD_SOC
- PD_CORTEX
- PD_CPU
- PD_L2

- Processors/Clusters
- Multi-Cores
- Caches
- Interconnects: PCIe, I3C, SafeSPI
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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>RESET Enable/Disable</strong>&lt;br&gt; L1: DBG1RSTDISABLE&lt;br&gt; L2: RSTDISABLE&lt;br&gt;L1/2 Reset Disable State=0 (Initial) to enable L1 Reset in PD</td>
</tr>
<tr>
<td>2</td>
<td><strong>Disable Data Cache</strong>&lt;br&gt; SCITLR.C (C - cache enable)&lt;br&gt;HSCITLR.C (cache enable)&lt;br&gt;System Control Register Cache line bit C = 0</td>
</tr>
<tr>
<td>3</td>
<td><strong>Clean and invalidate cache</strong>&lt;br&gt; DCCISW.LEVEL =3f000(L1)&lt;br&gt; DCCISW.LEVEL =3f111(L2)&lt;br&gt;DCCISW (Data cache clean and Invalidate by set way)</td>
</tr>
<tr>
<td>4</td>
<td><strong>Clean and invalidate by Virtual Address</strong>&lt;br&gt; DCCIMVACS&lt;br&gt;L1 HxVA, L1UnVA, L1HxS, L1UnS, L1Hd, L1Un, L1s[VA, B, Prd], DCCISW_s set way, DCCIMVAC_s virtual addr Bored&lt;br&gt;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 test clean, Branch Predictor..</td>
</tr>
<tr>
<td>5</td>
<td><strong>Memory Model Feature Register (MMFR)</strong>&lt;br&gt; L1HxV, L1UnV, L1HxS, L1UnS, L1Hd, L1Un, L1s[VA, B, Prd], DCCISW_s set way, DCCIMVAC_s virtual addr Bored&lt;br&gt;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 test clean, Branch Predictor..</td>
</tr>
<tr>
<td>6</td>
<td><strong>Disable Data Coherency</strong>&lt;br&gt; CPUCTRL.SMEN&lt;br&gt;Low power retention state(CPU RETENTION CONTROL REGISTER Switch Mode Power supply Enable = 0 (Disable Data coherency))&lt;br&gt;the following conditions are met&lt;br&gt;Read no snoop control signal ARLOCKM=1(For ACE Interface).Inner/outer shareable Cache and FOR Load/store&lt;br&gt;Write No snoop Control signal AWLOCKM=1(For ACE Bus interface transactions)Inner/Outer write through. For load/store&lt;br&gt;Read no snoop control signal with Exc set high)(For CHI Bus Transaction Interface)&lt;br&gt;Write No snoop Control signal with Exc set high)(For CHI Bus Transaction Interface)&lt;br&gt;For non shareable cache operations&lt;br&gt;Shareable, Non Shareable(Inner and Outer) Power domain control signals with / without L3 Memory</td>
</tr>
<tr>
<td>7</td>
<td><strong>Bus Interface Configuration signals</strong>&lt;br&gt;</td>
</tr>
</tbody>
</table>
UVM Low Power ASM Routines

```c
// FIRST SOURCE CODE
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "svpi.h"

#include "AVMv6T2.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,
      standby_wfi, not_of_standby_wfi, standby_wfe,
      not_of_standby_wfe} power_standby_methods_e;

void disable_cache_func() {
  asm volatile (
    "mrs x0, SCTRL_EL3\n"
    "bic x0, x0, #(1 << 12)\n"
    "bic x0, x0, #(1 << 2)\n"
    "msr SCTRL_EL3, x0\n"
    "isb"
  );
}

#define CLEAN_INVALIDATE_DCACHE_MACRO(op) (\{
  asm("dmf ish"); /* ensure all prior inner-shareable accesses have been observed*/
  asm("mrs x0, CLEBRR_EL1"); \/* get 2 x level of coherence*/
  asm("and w3, w0, #0x7f000000"); 
  asm("lsl w3, w3, #23"); 
  asm("cbz w3, "#op" finished); 

  asm("mov w10, #0"); /* w10 = 2 x cache level*/
  asm("mov w0, w1"); /* w0 = constant 0x1*/
  asm("add w2, w10, w10, lsr #1"); /* calculate 3 x cache level*/
  asm("lsl w3, w0, w2"); /* extract 3-bit cache type for this level*/
  asm("and w3, w3, #0x7"); 
  asm("cbz w3, #2"); 
  asm("b.it "#op"_next-level"); /* no data or unified cache at this level*/
  asm("msr CASSR_EL1, x10"); /* select this cache level*/
  asm("isb"); /* synchronize change of cassel*/
  asm("mrs x1, CSSIDR_EL1"); /* read csside*/
  asm("add w2, w1, #7"); /* w2 = log2(1linelen)-4*/
  asm("add w2, w2, #4"); /* w2 = log2(1linelen)*/
  asm("subfx w0, w1, #1, w10"); /* w4 = max way number, right aligned*/
  asm("lcl 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, w9, w5"); /* w16 = amount to decrement way number per iteration*/
  
}
```

accellera SYSTEMS INITIATIVE
UVM Low Power DPI Package

```c
//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_cmos_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;
```
UVM Low Power Scenario Package

```verilog
//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,c5,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

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 instructions 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 propose 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?