Ace'ing the Verification of SOCs with Cache Coherent Heterogeneous Multiprocessors Targeted for Optimized Power Consumption

Mehul Kumar, Broadcom Inc., San Diego, CA (mehulkum@broadcom.com)
Tushar Mattu, Synopsys Inc., Marlboro, MA (Tushar.Mattu@synopsys.com)
Amir Nilipour, Synopsys Inc., San Diego CA (Amir.Nilipour@synopsys.com)

Abstract

In most of the electronic devices that we use today, the requirements to have higher processing capabilities with optimized power consumption has become a central theme. This is driven by a need for higher processing capabilities while continuously optimizing on power consumption is imperative in most of the electronic appliances that we use today. For such multi-processor intensive SoCs, there is a requirement to maintain coherency across the system (including peripherals and the main memory). This is critical to have multiple applications working together. Creating stimulus for cache coherent SoCs has become increasingly complex, with the requirement to create the correct random, yet coherent traffic, while additionally ensuring that the system remains coherent. It is also important to also ensure that the verification framework can validate that interconnect is able to maintain coherency or the fabric is able to handle all cache line transitions across the different processors in the SoC.

In this paper, we look at two specific aspects while dealing with these challenges. First, we see how we can leverage UVM based capabilities of AMBA 4 Coherency Extensions (ACE) based BFMs and Verification IPs to create the necessary traffic patterns. This would involve hierarchically building up existing sequences and nesting them together to create increasingly complex patterns. This would also involve building up the required configurability to manage the different requirements expected of this system. This would go together with the monitoring framework that would leverage the passive BFMs and components to monitor the increasingly complicated traffic across the fabric and verify not only the functional response but also dump out the relevant performance statistics. Subsequently, we would want to go down to the next level of abstraction where we look at how we can have a LP aware testbench with Power aware UVM agents drive the verification of low power requirements of our systems with multiple cache coherent buses.

1. INTRODUCTION

The ACE specification enables system-level cache coherency across clusters of multi-core processors. The verification of such a system poses significant challenges. When planning the functional verification of such a system, we need to have an effective testing strategy to ensure not only that all aspects of the protocol are tested, but also that bugs are caught with the least effort.

Our team has been involved in advanced verification of ARM based systems including AXI interconnect designs for a few years resulting in successful tapeouts. However, the addition of cache coherency and low power design requirements has instantly increased the verification complexity. Considering the immediate challenge of ramping up our team on the protocol itself and planning for an effective and timely sign-off, made us consider the well-

proven Synopsys VIP, enabling us to leverage its innovative features so we can solely focus on verifying the intricacies of our design.

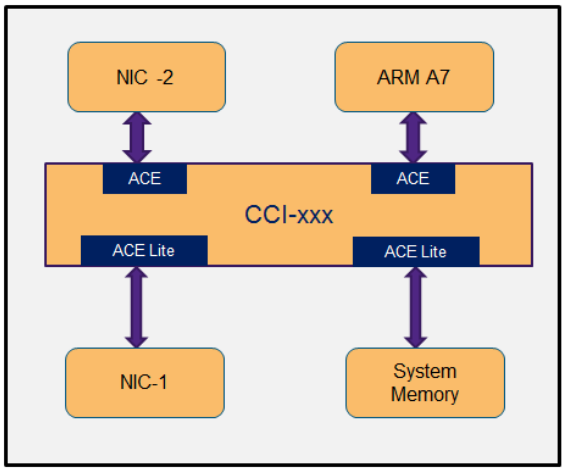

Figure 1 (Design Block a.k.a backplane)

In this paper, we will discuss a number of specific areas, where we faced most challenges in cache coherent and low power design verification and how we addressed them during our verification.

2. CACHE COHERENCY VERIFICATION

Our ultimate goal was to focus on verifying the coherency aspects of our design by fully leveraging various features of the VIP and not reinventing the wheel. These features span different areas appropriate to the task as hand and are described below.

2.A Configuration

This was most critical piece to start with, to ensure parameters in our VIP agents match those in the design. Some of these were easy to find based on experience but others were new such as:

- Address and Domain mapping
- Cache configuration matching with design caches
- DVM and Barrier etc.

Figure 2 illustrates the testbench topology that includes connections to both active and passive VIP agents over AXI protocol interfaces of design.

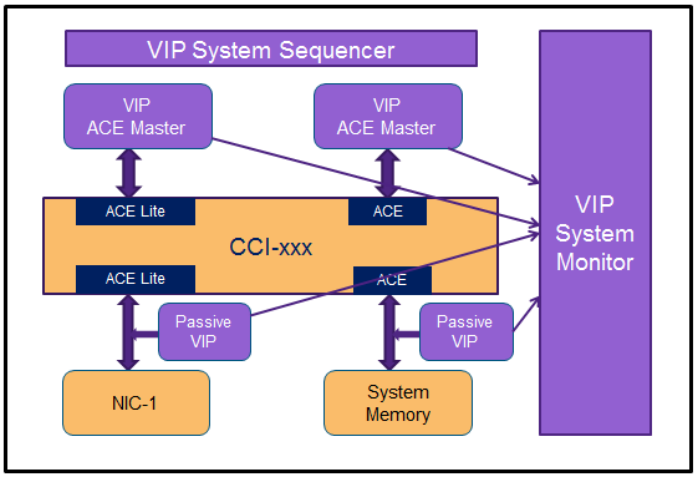

Figure 2 Testbench Topology with AXI-ACE VIP

Figure 3 shows some of the configuration objects that need to be considered for proper operation.

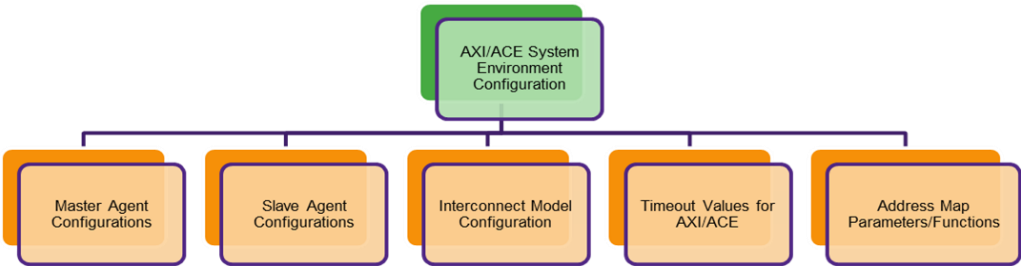

Figure 3 AXI/ACE Configuration Objects

Understanding what fields need to be configured along with all their possible values can be a cumbersome and error prone task. The VIP configuration creator (figure 4) came handy to provide drop down list of attributes with pre-populated values to be selected interactively. This method will result in valid configurations as tool implicitly validates its correctness per protocol.

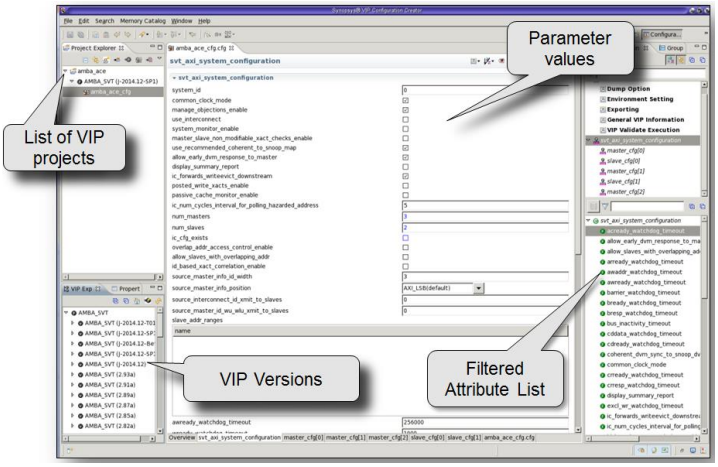

Figure 4 VIP Configuration Creator

However, one can always manually edit various configuration objects in the environment as shown in Figure 5.

```
class cust_system_configuration extends svt_axi_system_configuration;
    for(int i=0;i < `NUM_MASTERS ;i++) begin
     master_cfg[i].is_active = 1;
     master_cfg[i].axi_interface_type = svt_axi_port_configuration::AXI_ACE;
     master_cfg[i].snoop_data_width = 64;
     master_cfg[i].cache_line_size = 64;

  begin
    int inner_domain_masters_0;
     inner_domain_masters_0 = new[2];
     inner_domain_masters_0 = {0,1};
     void'(create_new_domain(0,svt_axi_system_domain_item::INNERSHAREABLE,inner_domain_masters_0));
    set_addr_for_domain(0,64'h100000000,64'h1fffffff);
..
endclass
```

Figure 5 Manual Edit of System Configuration

We still faced some initial hiccups with first run of the tests such as the following scenarios:

- Snoop control register not configured in design
- Mismatch in speculative read configuration compared to Design

As the following run time UVM error message shows, the inbuilt VIP system checkers helped to isolate the failures for mismatching configuration on cache line size versus number of bytes transferred in full cacheline transaction (CLEANINVALID).

```
UVM_ERROR :    6075 ns :
[register_fail:AMBA:AXI_ACE:cache_line_arsize_valid_check] Description: Monitor
Check that the total number of bytes transferred in the transaction is equal to
the cache_line_size! - cache_line_size = 2048 (bytes). burst_size =
BURST_SIZE_128BIT. burst_length = 4. coherent_xact_type = CLEANINVALID :
```

Figure 6 Messages from VIP System Monitor

2.B Stimulus and Coverage

Many types of predefined coherent sequences from VIP library can be registered with AXI or ACE component sequencers that exist in the AMBA System environment. Some sequences work at the master and slave agent port levels, while others work at the system virtual sequencer level where multiple master and slave agents are involved.

Generating correct and efficient stimulus from AXI masters is complex and involves careful consideration of many factors some of which are described below and shown in figure 7.

- All the legal initial states of the cache lines for various masters need to be generated and responses validated.
- As the problem space is further expanded with the final cache line state (after a transaction ends) being determined by various configuration options, all the response types for each initial cache line state need to be tested.
- Since accesses to overlapping addresses can be initiated, need to ensure that all the rules are correctly followed by the interconnect when two masters access the same/overlapping address.

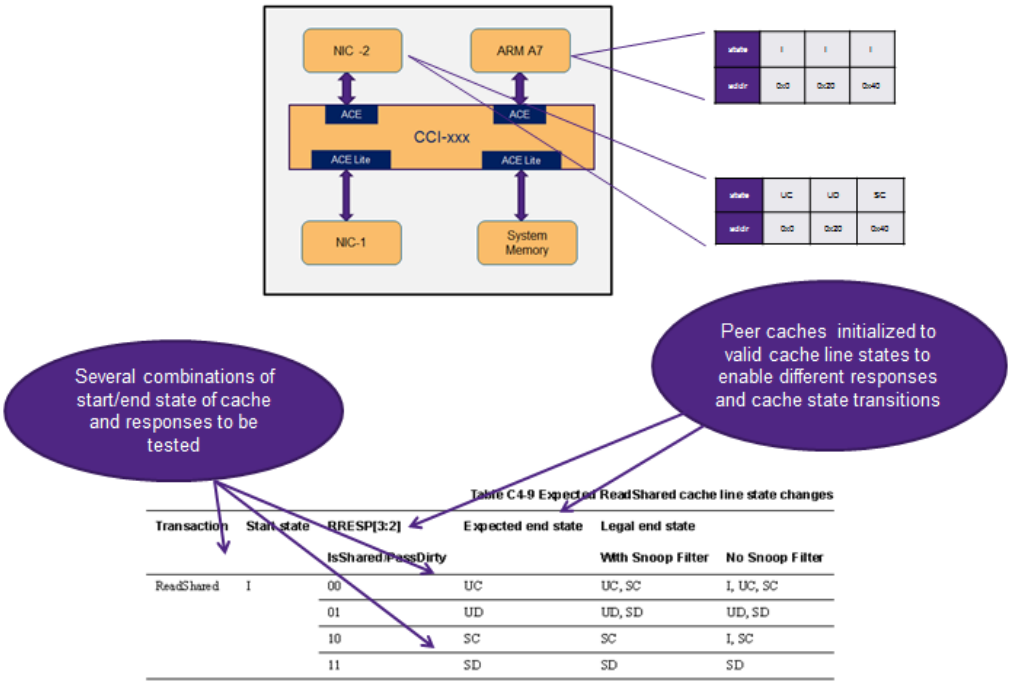

Table C4-9 Expected ReadShared cache line state changes

Transaction	Start state	BRESP[3:2] IsShared:PassDirty	Expected end state	Legal end state	
				With Snoop Filter	No Snoop Filter
ReadShared	I	00	UC	UC, SC	I, UC, SC
		01	UD	UD, SD	UD, SD
		10	SC	SC	I, SC
		11	SD	SD	SD

Figure 7 Cache Line Verification Complexity

Most of the above scenarios were covered by use of VIP test suite, which provided the capability to program required number of participating masters and to customize target address ranges etc. However , initially some of the desired sequences didn't exist as part of test suite, but were made available by Synopsys extracting from their internal regression suite.

Usage of these test suite sequence was as easy as to select the sequence and set it to VIP provided AXI system sequencer in **uvm_build_phase()** of a **uvm_test** inherited class as shown in figure 8 for system, master and slave sequencers respectively.

```
uvm_config_db#(uvm_object_wrapper)::set(this,
"env.axi_system_env.sequencer.main_phase", "default_sequence",
svt_axi_ace_master_two_port_overlapping_add_store_sequential_sequence::type_
```

```
uvm_config_db#(uvm_object_wrapper)::set(this,
"env.axi_system_env.master[0].sequencer.main_phase", "default_sequence",
svt_axi_basic_writeback_full_cacheline::type_id::get());
```

```
uvm_config_db#(uvm_object_wrapper)::set(this,
"env.axi_system_env.slave[0].sequencer.main_phase", "default_sequence",
svt_axi_slave_memory_suspend_response_sequence::type_id::get());
```

Figure 8 Using VIP Built-in Sequences

By setting specific weights on required kind of transactions within the VIP built-in coherent sequences, very high level of coverage was achieved per our coverage needs.

Figure 9 shows a how the sub-sequences of a hierarchical sequence initiate the required cache line initialization from peer ACE master port to put cache of targeted master port to one of the desired valid state, and create the required coherent transaction from that targeted master.

svt_axi_ace_master_two_port_overlapping_addr_store_sequential_sequence	Sends a set of concurrent, sequential store accesses from two ports to the same set of overlapping addresses. The store type transactions can be MAKEUNIQUE, READUNIQUE, CLEANUNIQUE, WRITEUNIQUE or WRITELINEUNIQUE based on the interface types of the ports and the weights. If first_port_cleanunique_wt or second_port_cleanunique_wt is not zero, cachelines are initialised since CLEANUNIQUE can be sent only from a cacheline in shared state. Only cachelines from which CLEANUNIQUE needs to be sent are initialized. The number of CLEANUNIQUE transactions sent are determined by the formula sequence_length*cleanunique_wt/(sum of weights of all xact types). Initialisation is done by sending MAKEUNIQUE transactions from one ACE port and READSHARED transactions from another ACE port to the same set of addresses. Snoop transactions for READSHARED type snoop are programmed (in the corresponding tests) to always assert svt_axi_snoop_transaction :: snoop_resp_datatransfer and svt_axi_snoop_transaction :: snoop_resp_isshared so that a shared state of the cacheline can be acheived in both masters. Once cachelines are initialised, sequential stores from first_port_id and second_port_id are made.

Figure 9 Example of a built-in hierarchical sequence

Leveraging VIP provided sub-sequences enabled us to create specific and custom scenarios to ensure coherency across caches and memory is not broken at any time.

VIP System Monitor was key factor to point out any system coherency violation, data integrity and transaction routing issues.

System Monitor also ensured that protocol specific ordering is followed, when there is concurrent accesses to same cacheline OR coherent and snoop happening at the same time.Even beyond that , errors flagged by System Monitor helped us clean up testbench issues like missing pin connection , configuration mismatch etc.

VIP also provided a robust coverage model, which we customized with combination of cover group enable switches and Verdi exclusion mechanism wherever finer granularity was needed. Customized covergroups were developed based on our design-testing requirement by copying and modifying similar ones from VIP open source documentation...

While there was need to ensure system wide coherency, there was another aspect to our design verification that whenever system goes to low power mode, all the testbench components need to align with that requirement and all initiated traffic and its associated snoops should be completed in all means from request to response. Using VIP provided APIs and events, we were able to query for transactions to conclude the traffic before initiating power down sequence. In the following section, we will detail the low power requirement for design verification and how it was achieved.

3. AXI LOW POWER VERIFICATION

3.A INTRODUCTION : CLOCK TREE, POWER RELATION

ASICs designed for applications like mobility, tablet computing, handheld devices etc. have a mandatory requirement for low power consumption to facilitate longer battery life. A common methodology adopted to reduce power consumption on SOC is clock gating. It is a technique to reduce the dynamic power consumption by turning off the sequential circuitry or flops during an idle state of operation. During this period of non-operation the circuit would gate off all clocks that are not needed anymore.

3.B SCHEMES TO CONSERVE CLOCK TREE POWER

In SoC designs clock gating may be done at two levels:

- Internal Clock Gating cells insertion: During synthesis, the tools identify groups of FFs which share a common enable control signal and use them to selectively switch off the clocks to those groups of flops.
- Implementing traffic aware clock controllers are another way of shutting off clock to entire logic cone. It stops the clocks for individual blocks when those blocks are inactive. Since large cones of logic are not switching for many cycles it saves substantial dynamic power. The simplest and most common form of clock gating is by use of "AND" to selectively disable the clock to individual blocks by a control signal.

This section would provide more detail on AXI based clock controllers to turn off clock for the backplane fabric.

Design clock gating Requirement from AXI controller perspective :-

1) Deep sleep Mode :
   This is mode where SoC enters ultra-power saving state thereby doing the following :
   a) Turning off all clocks , thereby using the C channel to request clock controllers to gate off clocks
   b) Assertion of all isolations and resets
   c) Turning off Bucks and LDOs to except always ON domain to conserve power
   d) Finally turning off Crystal clock thereby completing the power mode entry

   Wake up process was exactly same in reverse order , except that BUCKs are turned on before XTAL clock. This was verified in simulation using same AXI VIP agents for backplane clock gating with and without UPF. To run without UPF bench needed to emulate all isolations and power turn-off behavior using forces.

2) Dynamic Low Power Mode:

Figure.1 represents AXI based SoC architecture with multiple NICs and CCI controller. The design requirement for dynamically turning off backplane clocks when PCIe link is in L1.1 would help save power. Since the traffic is absent to-from IPs, SoC should be able to gate clocks to either individual AXI ports or entire backplane depending on the C-channel handshake. This would enable to save power on logic cone working on AXI port clock. The system clock controller could easily utilize the C-channel of AXI to gather control information of which interface is requesting clock to be turned on or off.

3.C AXI BASED C CHANNEL CLOCK GATING

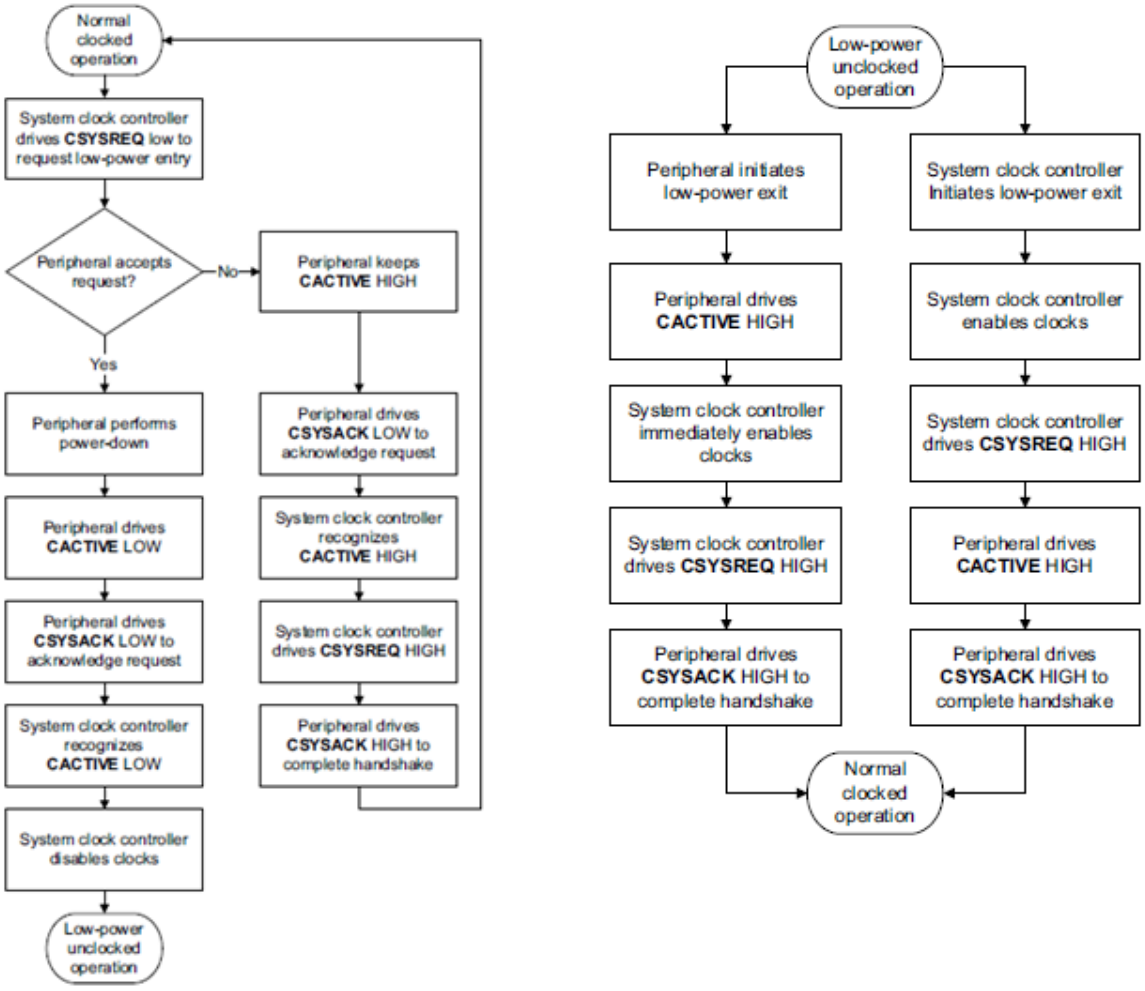

Figure 8 Typical flow for requesting entry(Left) and exit(Right) to low-power state

The low-power clock control interface consists of the following signals:

- A signal from the peripheral indicating when its clocks can be enabled or disabled
- Two handshake signals for the system clock controller to request exit or entry into a low-power state.
  The CACTIVE signal indicates whether the peripheral requires a clock signal. The peripheral asserts CACTIVE HIGH when it requires the clock to be enabled, and the system clock controller must enable the clock immediately.
  The peripheral deasserts CACTIVE to indicate that it does not require the clock.
  CSYSREQ The system clock controller uses the CSYSREQ signal to request:

- The peripheral enters a low-power state. The system clock controller drives the CSYSREQ signal LOW to initiate the request.
- The peripheral exits a low-power state. The system clock controller drives the CSYSREQ signal HIGH to initiate the request.
  CSYSACK The peripheral uses the CSYSACK signal to acknowledge:
- The request to enter the low-power state. It drives CSYSACK LOW when it recognizes this request.
- The request to exit from low-power state. It drives CSYSACK HIGH when it recognizes this request.

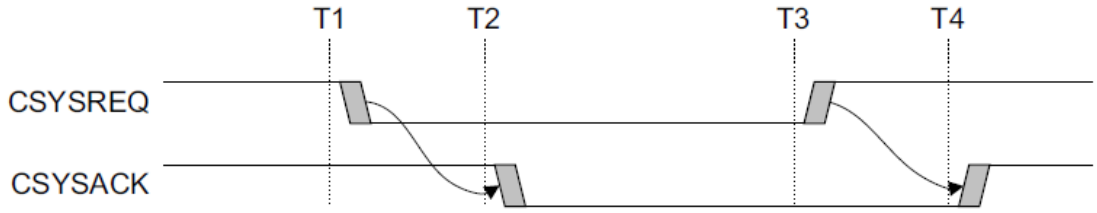

Figure 10

3.D CHALLENGES & MOTIVATION

To verify dynamic AXI based clock gating scheme requires interface level monitoring of C-Channel and overall scoreboard which could capture port states published by individual port monitors and use that information to implement overall abstract checks. Such implementation would need maintenance and since it is part of AXI specification, it became a motivation to collaborate with Synopsys to come up with C-Channel interface level monitor/checker as part of SVT AXI package. This would alleviate maintenance, compliance to AXI specifications and standardization issues. Only customer specific implementation would be an end-end to scoreboarding depending on SoC clock controller behavior based on the c-channel handshake information being published by SVT-AXI monitors.

Here are examples of many such useful transaction field provided by VIP monitor to build custom scoreboard:

- Whether the object is related to power down handshake or power up handshake.
- Initiator of the powedown/powerup – Either peripheral or clock controller
- Number of cycles it took to complete the handshake.
- Number of clock cycle delay between different signals (cactive to csysreq; csysreq to csysack; etc)

Below are some of design issues which were caught by the combination of custom scoreboard and VIP checker

- Outstanding responses when the controllers request the clock turn off, therefore dropping responses. (scoreboard)
  e.g:
  Clock not turning on:
  UVM_ERROR @ 118555.1ns: [scoreboard] csysack_cd_clk_sysm not changing as expected

  Clock not turning off after required N clock cycle expiry:
  UVM_ERROR @ 119473.2ns: [scoreboard] cactive_cd_clk_sysm csysreq_cd_clk_sysm not changing as expected after Timer timeout

- Error response needing IPs to take corrective action, whereas the gating logic requests clocks to be turned off. The new request would wake up the controller however, it would incur latency penalty on retries every time clock is shut off. (scoreboard)

- Clocks are never turned off even when the interface remains idle for required amount of time (VIP check)
- C-channel handshake protocol signaling adherence (VIP check)
- Requests from IPs do not result in clocks to be turned on. (VIP check)

3.E STEPS TO INTEGRATE AXI SVT LOW POWER AGENTS

SVT AXI LP Monitor

Module Top

- svt_axi_if should be present in the module top.

```
svt_axi_if axi_if();
```

The 'svt_axi_if' interface now also contains an array of low power interfaces, lp_if[] in addition to master_if[] and slave_if[].

- Connect clock, reset and low power signals to lp_if. Signals of low power interface are aclk, aresetn, cactive, csysreq and csysack

```
assign axi_if.lp_if[0].aclk      = clk;
assign axi_if.lp_if[0].aresetn = rstn;
assign axi_if.lp_if[0].cactive =  cactive;
assign axi_if.lp_if[0].csysreq = csysreq;
assign axi_if.lp_if[0].csysack = csysack;
```

System Configuration

Low power configuration is required in the system configuration file with following:
Configuring the number of low power masters using the attribute **num_lp_masters**

```
this.num_lp_masters = 1; //This needs to be configured before create_sub_cfgs()
```

Enable/disable low power protocol checks using **protocol_checks_enable**. This is enabled by default.
Low power master monitors have '**item_observed_port** which collects and write low power transactions whenever low power activity is observed on the bus.

```
axi_system_env.lp_master[0].monitor.item_observed_port.connect(lp_listener.analysis
_export);
```

Example class snippet is given below. Master and slave configurations are not included.

```
class cust_svt_axi_system_configuration extends svt_axi_system_configuration;
  function new (string name = "cust_svt_axi_system_configuration");
    super.new(name);
    /** Assign the necessary configuration parameters.
      */
   this.num_masters = 1;
   this.num_slaves  = 1;
   this.num_lp_masters = 1;
   this.lp_master_cfg[0].protocol_checks_enable = 1'b1;
endfunction
endclass
```

System level scoreboard

- Provide write implementation for the individual analysis ports of each LP agent
- Capture the C-channel information published as below:

```
UVM_INFO ./env/axi_basic_env.sv(45) @ 490000: uvm_test_top.env.lp_listener [lp_listener] inside
write method
----------------------------------------------------------------------
Name                            Type                    Size  Value
----------------------------------------------------------------------
lp_entry_obj                    svt_axi_service         -     @1749
  causal_xact                   object                  -     <null>
  implementation                da(object)              0     -
  original_xact                 object                  -     <null>
  trace                         da(object)              0     -
  lp_entry_active_req_delay     real                    64    185.000000
  lp_entry_req_ack_delay        real                    64    115.000000
  lp_exit_prp_active_req_delay  real                    64    0.000000
  lp_exit_prp_req_ack_delay     real                    64    0.000000
  lp_exit_ctrl_req_active_delay real                    64    0.000000
  lp_exit_ctrl_req_ack_delay    real                    64    0.000000
  lp_exit_ctrl_active_ack_delay real                    64    0.000000
  lp_handshake_type             lp_handshake_type_enum  32    POWER_DOWN
  lp_initiator                  lp_initiator_type_enum  32    PERIPHERAL
  lp_active_assertion_time      real                    64    190.000000
  lp_req_assertion_time         real                    64    375.000000
  lp_ack_assertion_time         real                    64    490.000000
  begin_time                    time                    64    190000
  end_time                      time                    64    490000
----------------------------------------------------------------------
```

- Implement clock controller behavior
- Capture port clock state and flag any errors in case the clocks are not turned off/on

4. CONCLUSION

The verification performed with combination of Synopsys-SVT and end-end Scoreboard lowered risk and provided necessary confidence on RTL implementation of complex dynamic clock gating scheme deployed in the SOC. The design issues are caught early in the RTL design cycle using the methodology proposed. Initially the verification effort sounded complex with additional complexity of Coherency protocol (AXI-ACE) and Low power design aspect. However due to the capabilities and ease of integration provided by Synopsys VIP , the bulk of verification cycle was spent in writing design specific tests, checks and coverage.