INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 5, ISSUE 9, SEPTEMBER-2014
ISSN 2229-5518

Implementation of Open Core Protocol transaction Verification IP using System Verilog UVM methodology

Siddaram Patil, Arun Kumar, Dr.v.Venkateswarlu

Abstract - This paper presents Implementation of the Reusable Open Core protocol (OCP) transaction Verification IP (VIP) using universal verification methodology (UVM) and very easy to implement the functional coverage and SystemVerilog assertions (SVA) using powerful System Verilog (SV) Language. As verification of the design has become most challenging task, verification is important factor for achieving time to market (TTM) of the product. This paper address briefly on how the UVM methodology and SV enables developing VIP easier and quickly over traditional Hardware description language(HDL) verilog or VHDL and Hardware verification language(HVL) Specman.

Keywords:
OCP, SV, UVM, SVA, VIP, SOC, TTM, HDL, OOPS

I. APPLICATION OF SV AND UVM IN OCP VIP.

The Current VLSI industry is moving rapidly in adapting the feature of SV and UVM in their IP and SOC level verification because of its strong advantages over the traditional HDL language like Verilog or VHDL and HVL language specman.

As the system Verilog is developed based on OOP's (object oriented programming) concepts and superset of Verilog, the environment can be extended and add new features without affecting the original existing code. SV supports constrained randomization, assertion and functional coverage.

UVM methodology reduces the time to develop VIP by using already built in base classes for all the required components from the UVM library and also helps re-using the VIP environment at different level of abstraction i.e. at module level and chip level.

Siddaram Patil, M Tech, Final Semester VLSI Design and Embedded Systems, VTU Extension Centre, UTL Technologies Bangalore -22, Karnataka, India.
srpatilsedam@gmail.com

Arun kumar, Professor,VTU Extension Centre,
UTL Technologies Bangalore -22, Karnataka, India,
arunkumar@utltraining.com

Application of SV in OCP VIP is that, the SV interface construct is used for communicating class based environment to module based environment and creating array of physical interface which helps in generating multiple OCP master/ slave VIP interface without much effort. SV constructs cover group, cover point and covergroup.sample is used for achieving functional coverage. SV construct property and assert is used for assertion coverage. The interface, functional coverage and assertion blocks shown separately in fig 1

Application of UVM in OCP VIP is that, the UVM built in base class for different components are used from the UVM library to develop OCP VIP such as, uvm_env class to build the environment, uvm_test class to develop the tests, uvm_sequence_item class to build properties of base tests, uvm_sequence class to develop the different test scenario, uvm_driver to generate the transactions to DUT for OCP_master/ slave driver, uvm_monitor to sample the activity of OCP bus

.

OCP VIP utilized other features of UVM like uvm_config_db::set to configure the physical interfaces or array of physical interface to different components in the environment such as OCP master/ slave agents, master/ slave driver, master/ slave sequence. UVM IS_ACTIVE feature can be used to make the OCP_VIP agent as active or passive based on value of IS_ACTIVE bit, this feature enables the OCP VIP agent to contain driver, sequencer and monitor if UVM IS_ACTIVE is set otherwise only monitor is enabled. Different OCP components are shown in fig 1.

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 5, ISSUE 9, SEPTEMBER-2014
ISSN 2229-5518

II. INTRODUCTION

Open core transaction protocol is data/ signal communication protocol in on chip core interfaces or in SOC. OCP data communication models range from simple request grant handshaking through pipelined request – response to complex out –of order transaction.

OCP describes a point-to- point interface between two communication module, such as IP cores and bus interface modules(bus wrappers)[1].in SOC the more number of IP core behavior, performance and interface requirements, a standard fixed definition interface protocol cannot address the all the system interface requirements. The need to support test requirement and verification adds a more complexity to the interface. To address the all system interface requirement, OCP defines a highly flexible and configurable interface.

As OCP makes IP modules independent of integration environment and interconnection technology, helps in reuse of OCP complaint modules in multiple SOC products.

The OCP VIP is a ready-made, highly configurable UVM verification environment, suitable for design under test (DUT) with an OCP interface. The OCP VIP can generate stimuli in an OCP bus format (single and burst), execute traffic over a bus as an OCP master to a DUT slave. Respond, as an OCP slave, to traffic coming from a DUT master. Ensure that the DUT adheres to OCP protocol rules, collect bus traffic and DUT behaviour information, according to OCP coverage items, all UVC behaviour is in accordance with the OCP 3.0 Specification.

The paper is organized as follows: section III describes the OCP VIP architecture. The Basic OCP signal description table in section IV. Flow Charts for master slave interaction and basic write read in section V. Implementation of assertion and functional coverage in VI. Typical application of OCP VIP in VII. The simulation results are presented in section VIII. Re-using of OCP VIP to generate multiple OCP master and slave interfaces in VIII and the conclusion is arrived at section XI.

III. OCP VIP Architecture in System Verilog UVM

The above fig 1 shows the OCP VIP architecture using System Verilog UVM methodology, it contains master and slave agents are instantiated in the OCP environment. Each master and slave agent contains its own monitor, Sequencer and driver respectively.

OCP Agent can be configured as Passive or active based on the requirement. The configuration can specify the agent as passive, which disables the creation of sequencer and driver. OCP configuration decides the how the OCP environment should function.

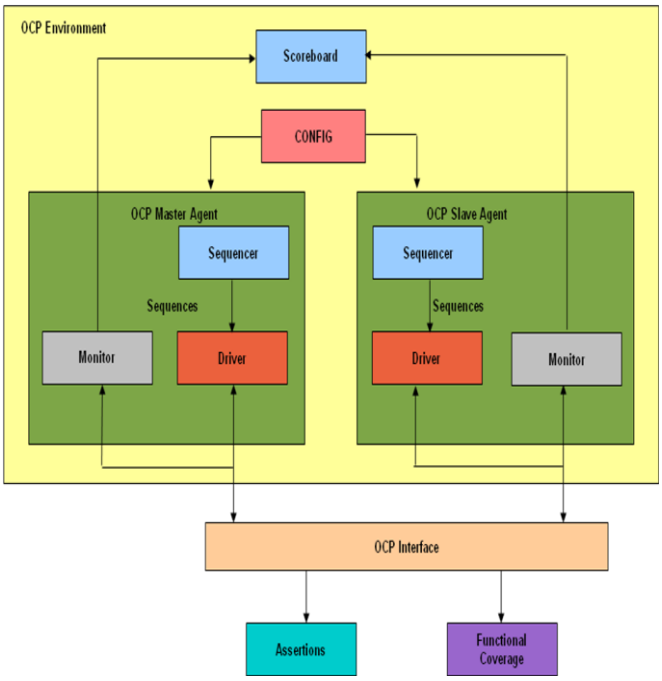


Fig 1 OCP verification IP architecture block diagram

The OCP Sequencer supports Generating and driving bus traffic as an OCP master. The UVC emulates the full behaviour of an unlimited number of OCP masters capable of generating all types of OCP transfers. Responding to bus traffic as an OCP slave. The UVC emulates the full behaviour of an unlimited number of OCP slaves that respond to traffic over a bus and generates all types of responses to a DUT master.

The OCP Driver support all OCP protocol data and address widths, OCP protocol burst transfers like continues READ and continues WRITE and assertions and checks for protocol violations.

The OCP monitor operates in three different modes: Bus Monitor, Master Monitor and Slave Monitor. By default Bus Monitor is active all the time in OCP environment. The UVC logs the bus traffic for the purpose of debugging its elements and DUT devices. Provides option for data integrity check, coverage collection. Supports all OCP Single, Burst, Tags and Treads Operations and all the commands.

OCP Scoreboard takes transaction packets from master monitor and slave monitor and compares both of them, two queues of packets are implemented in the scoreboard in order to receive the packets from both the monitors and store them temporarily. As soon as both the queues become non-empty, one packet from each of the queues is pulled out and compared. If a mismatch occurs, error is given out.

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 5, ISSUE 9, SEPTEMBER-2014
ISSN 2229-5518

IV. BASIC OCP SIGNAL DESCRIPTION [11]

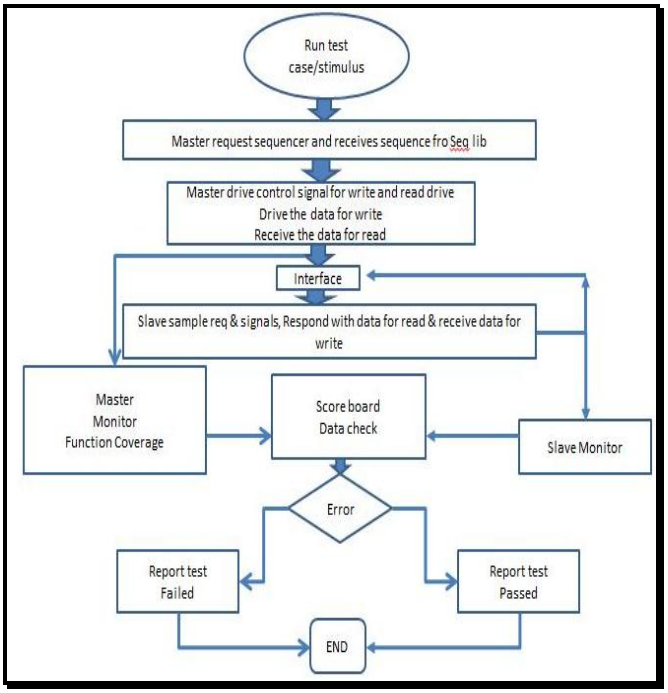
Name	Width	Driver	Function
CLK	1	Varies	Clock input
MAddr	Configurable	Master	Address
MCmd	3	Master	Command
MData	Configurable	Master	Write data
MDataValid	1	Master	Write data valid
MRespAccept	1	Master	Master accepts response
SCmdAccept	1	Slave	Slave accepts transfer
SDataAccept	1	Slave	Slave accepts write data
SData	Configurable	Slave	Read data
SResp	2	Slave	Transfer response

Table 1 Basic OCP signals

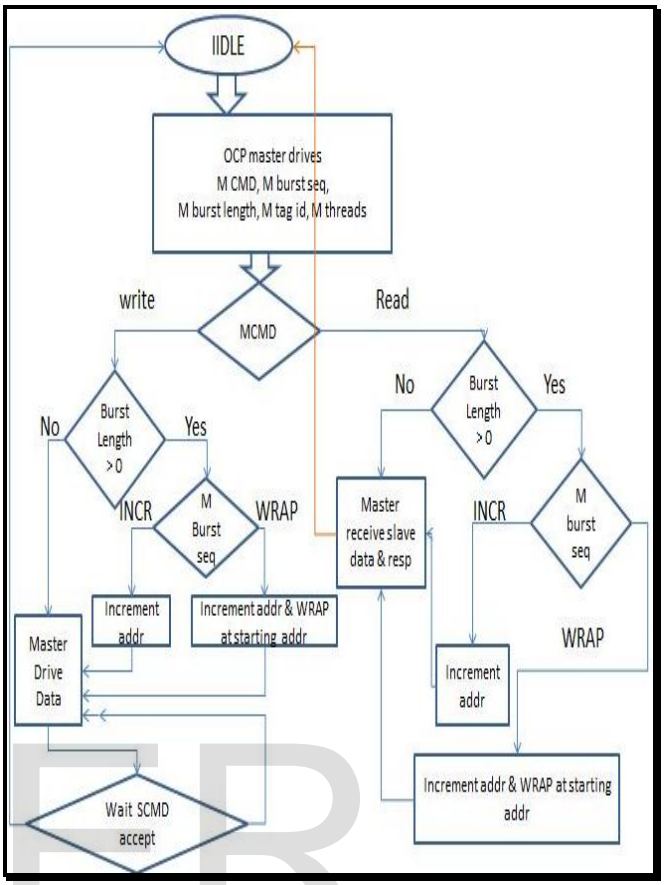
V. FLOW CHARTS

A. Master and Slave interaction Flow

This Flow chart explains how the master drives stimulus to slave, how slave responds for the master commands. Shows the monitors and scoreboard with test status PASS or FAIL.



B. OCP Write and Read Flow



VI. ASSERTION and FUNCTIONAL COVERAGE

A. Assertion [3] [11]

System Verilog Assertion helps in checking the behaviour of the code and it can be reused across the project without many difficulties. We can instantiate the assertion code in the test bench using "bind" command that will bind the assertion module ports to the DUT ports. Even we can insert the assertion in the interface. If you know the ports signals of modules, one can easily write the assertion logic independently without any dependency. Assertions also called as control oriented coverage

Example: when the SCmdAccept is low and transfer is not IDLE the Master should hold the data until the SCmdAccept is high. (Need to constraint the number of clock cycles to hold the data)

```
module assertions (ocp_interface ocp_vif); / /  {
property p1;
@(posedge ocp_vif.Clk)
disable iff (ocp_vif.MReset_n==1'b0 | |  flag == 0)
 (ocp_vif.SCmdAccept==0      &&      ocp_vif.MCmd!=3'd0)
          | -> ##1 $stable (ocp_vif.MAddr);endproperty
```

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 5, ISSUE 9, SEPTEMBER-2014
ISSN 2229-5518

assert property (p1) else $error("ASSERTION ERR:MAddr is not stable incase of SCmdAccept is low and transfer is not IDLE\ n\ n");
endmodule

B. Functional Coverage [2] [11]

Functional Coverage used to measure how effectively The VIP tested all the feature of design, functional coverage also called data oriented coverage.

Example:

```
class ocp_bus_monitor extends svm_monitor #(ocp_master_basic_sequence_item,ocp_slave_packet);
/-OCPFunctional Coverage groups Declarations

 covergroup ocp_cov ;
  option.per_instance = 1;
  //OCP Cover Points//////////////////////
  transTypes1: coverpoint intf.MBlockHeight {
               ignore_bins Low = {0,1};
               bins High = {2,3};}
  transTypes2: coverpoint intf.MBurstLength {
               bins MBurstLength_2 = {2};
               bins MBurstLength_4 = {4};
               bins MBurstLength_8 = {8};
               bins MBurstLength_16 = {16};
               bins MBurstLength_32 = {32};
               bins MBurstLength_64 = {64};}
  transTypes3:coverpoint intf.MBurstSingleReq;
  transTypes4:coverpoint intf.MCmd;
endclass
```

VII. TYPICAL APPLICATION OF OCP VIP

OCP VIP used to perform DUT verification at the block level or chip level. These features are enabled by configuration file.

A. Block-Level DUT Verification

The OCP VIP is typically used to verify individual bus agents, such as bus masters or bus slaves. Figure 2 and Figure 3 depicts typical applications of the VIP at block level.

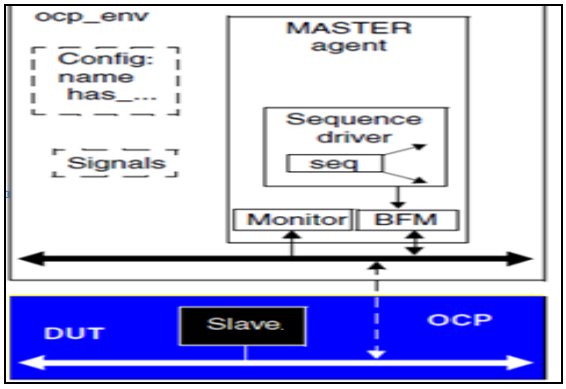


Figure 2 OCP VIP as Master-Testing a Single DUT Slave

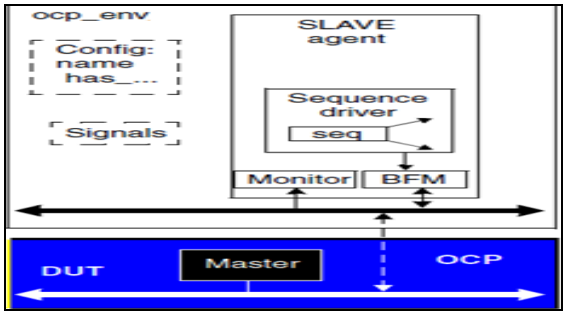


Figure 3 OCP VIP as Slave- Testing a Single DUT Master

B. Chip-Level DUT Verification

OCP Monitor can be used to monitor a DUT with Master and Slave. Figure 4 depict typical application of Bus monitor or Master monitor or Slave monitor

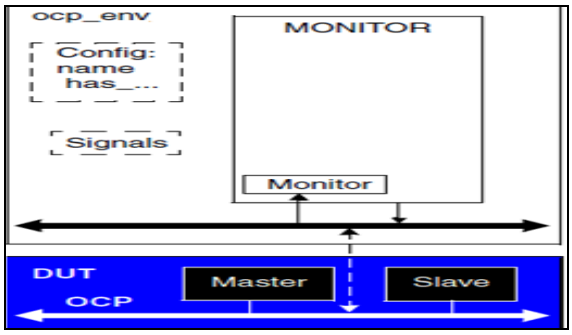
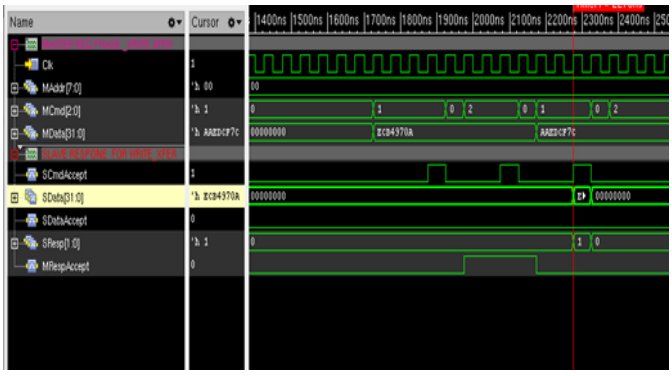


Figure 4 OCP Bus monitors- Monitoring a DUT with Master and Slave

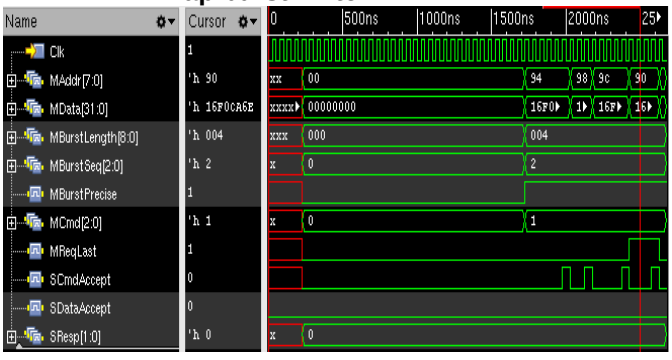
VIII. SIMULATION RESULTS

A. OCP BASIC TRANSACTION WAVEFORM

I. Read transfer

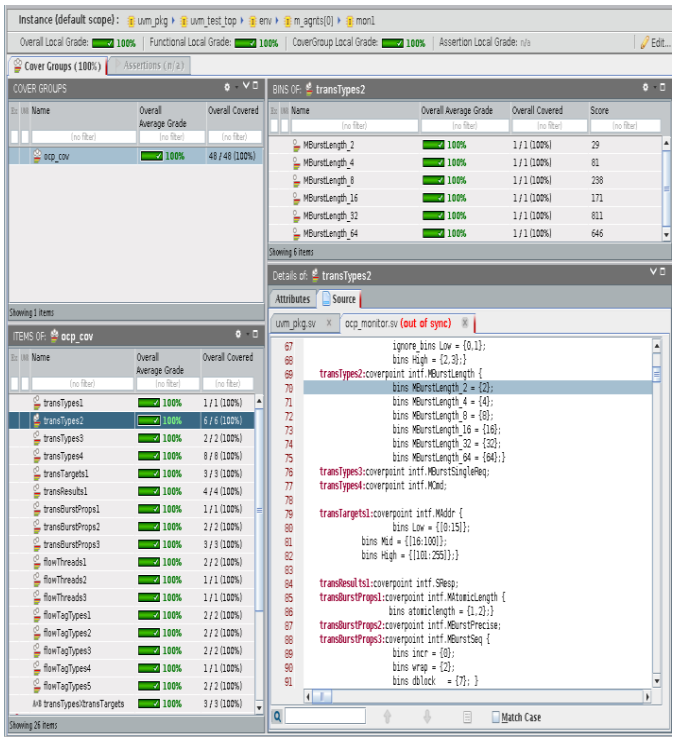


II. Wrap burst write

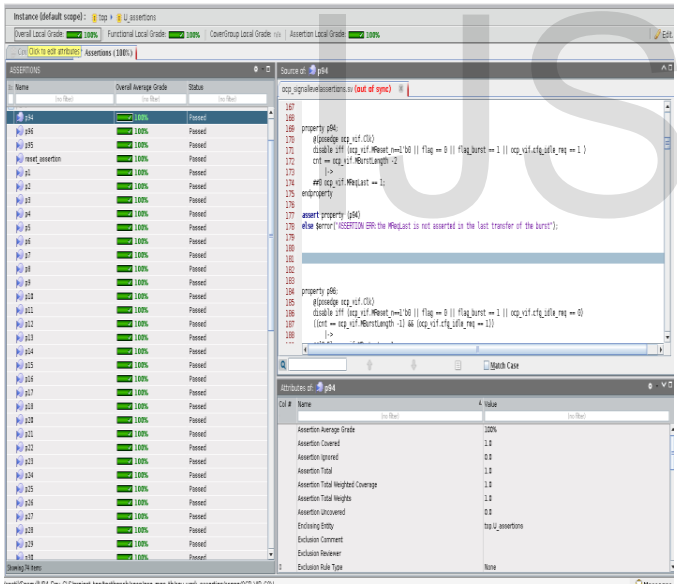


INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 5, ISSUE 9, SEPTEMBER-2014
ISSN 2229-5518

B. FUNCTIONAL COVERAGE REPORT



C. ASSERTION COVERAGE REPORT



IX. REUSE OCP VIP TO GENERATE MULTIPLE MASTER/SLAVE

Re-used the OCP VIP environment to create multiple master and multiple slave with the help of System verilog interface concepts, which is used to creates array of interface structure and also similarly UVM feature uvm_config_db::set method is used to assign each individual array of physical interface from the top level environment to array of master/ slave agents without any difficulties, the connections happens automatically.

The above implementation can be done in simple way as shown below steps.

1. Define how many number of master or slave instance required using define NUM_INTF, the physical OCP interface will be created NUM_INTF times in top level module environment

```
`define NUM_INTF 5
module top();
 reg Clk;
ocp_interface ocp [`NUM_INTF] (Clk);
 endmodule
```

2. Assigning the sequences to array of master/ slave, array created is using * i.e. env.m_agnts*/ env.s_agnts*. if we not use array structure, then individually we need configure NUM_INTF times(it will not be generic)

```
for (int k=0; k<`NUM_INTF; k=k+1)
 begin //mstr/slave
uvm_config_db#(uvm_object_wrapper)::set(this,
env.m_agnts*.sequencer.main_phase", default_sequence",
ocp_wrap_write_burst_seq::type_id::get());
uvm_config_db#(uvm_object_wrapper):set(this,
"env.s_agnts*.s_sequencer.main_phase","default_sequence
", simple_response_seq::type_id::get());
end
```

3. Connect the array of physical interface from top level to array of OCP master/ slave agents components without much effort using UVM feature uvm_config_db::set

```
for ( genvar k=1; k<`NUM_INTF; k=k+1) begin
initial begin
swrite(m_agnt_name,"*.env.m_agnts[%0d]", k);
$swrite(s_agnt_name,"*.env.s_agnts[%0d]", k);
uvm_config_db#(virtual
ocp_interface)::set(null,m_agnt_name,"intf",top.ocp[k]);
uvm_config_db#(virtual
ocp_interface)::set(null,s_agnt_name,"intf",top.ocp[k]);
uvm_config_db#(virtual
ocp_interface)::set(null,{m_agnt_name,".mon1"},"intf",to
p.ocp[k]);
uvm_config_db#(virtual
ocp_interface)::set(null,{s_agnt_name,".s_monitor"},"intf"
,top.ocp[k]);
uvm_config_db#(virtual
ocp_interface)::set(null,{m_agnt_name,".m_basic_s
equence_driver"},"intf",top.ocp[k]);
uvm_config_db#(virtual
ocp_interface)::set(null,{s_agnt_name,".s_driver"},"int
f",top.ocp[k]);end
    end
```

By changing only one parameter value NUM_INTF=10, environment has created 10 OCP master/ slave interface and observed transaction on all the master/ slave interfaces.

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 5, ISSUE 9, SEPTEMBER-2014
ISSN 2229-5518

X. CONCLUSION

System Verilog UVM methodology verification IP provides reusable and configurable VIP at Various levels of abstractions. Verification IP of OCP allows the customer to reuse the VIP environment in their soc environment without much effort when connecting multiple master/ slave VIP if the DUT contains multiple OCP master/ slave. Usage of Powerful feature of System Verilog assertion and functional coverage gives the confidence for designer to release the products. Reusing of Verification IP of OCP helps to achieve time to market of the SOC design

REFERENCES

[1] Chih-Wea Wang, Chi-Shao Lai, Chi-Feng Wu, Shih-Arn, and Ying-His Lin, "On-Chip Interconnection Design and SoC Integration with OCP", 13 Jun 2008, VLSI Design Automation and Test (VLSI-DAT)

[2] Shihua Zhang, Asif Iqbal Ahmed and Otmane Ait Mohamed, "A Reusable verification Framework of Open Core Protocol", Circuits and Systems and TAISA Conference, 2009, pp. 1-4, june 28, 2009

[3] Kun Tong and Jinian Bian, "Assertion-based Performance Analysis for OCP Systems."In Proc. Circuits, Signals, and Systems (CSS 2007), Banff, Alberta, Canada, July 2007

[4] Natale Barsotti, Riccardo Mariani, Matteo Martinelli, and Mario Pasquariello, "Dynamic verification of OCP-based SoC", in Proc. IEEE Int'l Conf. on System-on-Chip, Tampere, Finland, Nov. 2005, p. 22

[5] Chih-Wea Wang, Chi-Shao Lai, Chi-Feng Wu, Shih-Arn Hwang, and Ying-Hsi Lin, , "On-chip Interconnection Design and SoC Integration with OCP", Proceedings of VLSI-DAT, 2008, pp. 25-28, April 2008

[6] Chin-Yao Chang, Yi-Jiun Chang, Kuen-Jong Lee, Jen-Chieh Yeh, Shih-Yin Lin and Jui-Liang Ma, "Design of On-Chip Bus with OCP Interface", 28 Jun 2010, VLSI Design Automation and Test (VLSI-DAT)

[7] Chien-Chun (Joe) Chou, Konstantinos Aisopos, David Lau, Yasuhiko Kurosawa and D. N. (Jay) Jayasimha, "Using OCP and Coherence Extensions to Support System-Level Cache Coherence", Technical Paper, pg. nos.10, April 2009

[8] Bhakthavatchalu R, Deepthy G.R., Shanooja S, "Implementation of reconfigurable Open Core Protocol compliant memory system using VHDL" Industrial and Information Systems (ICIIS), 2010 International Conference, 2010, pp.213-218, July 29, 2011

[9] OCP-IP, *A SystemC OCP Transaction Level Communication Channel V2.2, 2006*

[10] Open Core Protocol (OCP) Specification, http:/ / www.ocpip.org/ home.

[11] Open Core Protocol Specification 3.0", International Partnership, 2000- 2009 OCP-IP Association, Document Revision 1.0.

[12] UVM Cook book http:/ / verificationacademy.com/ uvm-vm/ pdf?pdf=Cookbook%2FDac2011