Impact Factor (SJIF): 4.542

International Journal of Advance Research in Engineering, Science & Technology

e-ISSN: 2393-9877, p-ISSN: 2394-2444

Volume 4, Issue 5, May-2017

Development of Verification IP for DDR2 memory with performance monitor

Gorwadia Sumit[1], Mr. Jayesh Popat[2]

PG student [VLSI system design], Elect. & Comm. department., MEFGI, Rajkot, Gujarat[1]
Assist. Prof., Elect. & Comm. department., MEFGI, Rajkot, Gujarat[2]

Abstract—The goal of function verification is to find the errors in the design given by the engineers and to check the functionality of that design whether it give the expected output if not then change the design according to it to get desired functionality of DUT (design under test). This paper show the latest approach to meet above requirement using the UVM (Universal Verification Methodology).

Keywords—UVM, Verification, Environment, Verification IP, memory verification, System Verilog,

I. INTRODUCTION

The Universal Verification Methodology (UVM) is a standard verification methodology from the Accellera Systems Initiative that was developed by the verification community for the verification community. UVM represents the latest advancements in verification technology and is designed to enable creation of robust, reusable, interoperable verification IP and test bench components. The UVM gives a System Verilog base class library and rules. The UVM class library makes an unmistakable separation between the sequence that create stimulus and the structure which build verification environment. Client or user can set up test bench and produce stimulus utilizing the UVM base classes.

UVM is a methodology for SoC functional verification that uses TLM standards for communication between blocks and SV for its language or in other words it uses SV for creating components and TLM for interconnect between components. Methodology is basically set of base class library which we can use to build our test benches. UVM main goals are: reusability to reduce time to market and it is targeted to verify system from small to large concepts.

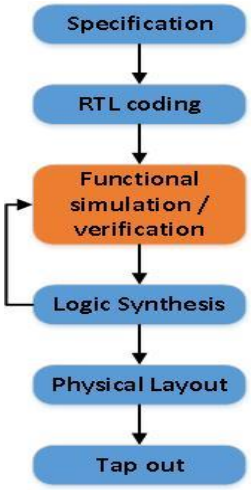

Figure 1. Verification flow of ASIC.

II. NEED OF UVM

Due to advancement in fabrication technology, more and more logic is being placed on single silicon die. Now, more than 70% time is spent on verification so there is need for construction a reusable and robust verification environment. Universal Verification Methodology was introduced to fulfill that goal. Also System Verilog is the only industry standard hardware verification language which is supported by all three largest EDA vendors. But System Verilog is still not sufficient to verify the design as it is remains under-specified as a language hence the UVM come to picture. System Verilog and UVM now form a virtuous circle.

The class based System Verilog features that support constrained random verification are sufficiently well defined and well implemented to allow the development of robust and portable verification class libraries, and the widespread use of those libraries ensures the ongoing support of the necessary language features by the tool vendors.

2.1 UVM Phases

In UVM, all the component implements the same set of phases which are run in predefined phases during simulation. Phase method is needed in order to synchronize the behavior of each component. The standards phases are as follows:

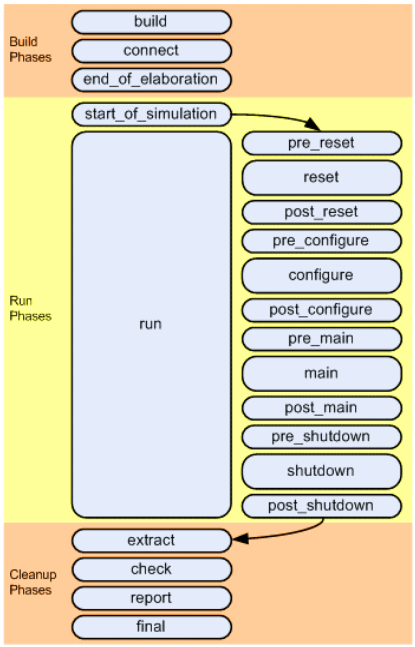

Figure 2. UVM Phases

2.1.1 Build:
It will create the child component instances as well as parent one.

2.1.2 Connect:
It is used to connect ports and exports on the child components. The connection can be ports to exports, exports to ports or ports to ports.

2.1.3 End of elaboration:
This phase makes the environment aware of the address range to which the slave agent should respond. It also means that verification environment has been completely assembled.

2.1.4 Start of simulation:
It will just notify the DUT that verification environment is configured and ready to simulate.

2.1.5 Run:
It runs the simulation. As shown in fig. 3, run phase is divided into several run phases. All the phases are represented by a function which run in zero simulation time except run phase hence task is used to define run phase which consume time.

2.1.6 Extract Phase:
It is post processing component. It is used to extract data from different point of verification and scoreboard.

2.1.7 Check Phase:
It is post processing phase used to check whether any unexpected condition meet in verification environment.

2.1.8 Report:

It is also post processing phase. It will just generate the report of test.

2.1.9 Final phase:

It shows that all the phases of UVM are completed and terminated the simulation.

III. TESTBENCH ARCHITECTURE

The subsequent section describes the aspect of verification additives:

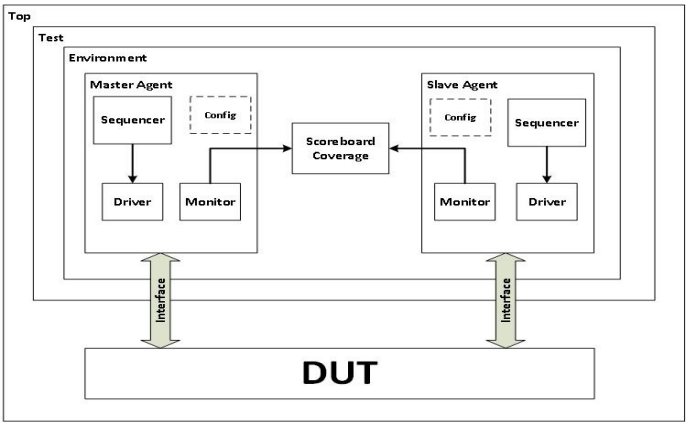

FIGURE 3. Verification Environment

3.1 Transaction

Transaction are basically the input parameter to the DUT. All of the transfer achieved among one of a kind component is executed through transaction [2]. In comparison to VHDL indicators and Verilog wires, transaction represent communique at an abstract level [3].

3.2 Driver

As the name endorse driver drive the dut indicators/signals. It essentially receives the transaction object from collection and converts into the pin interest [2]. The transaction from its collection and controls the signal-level interface to the DUT [3].

3.3 Sequence

All the series will run on sequencer components. The series of transaction carried out to the dut to test its conduct. Now whenever the sequencer demands the series of transaction, it's going to generate and carried out to the driver [2].

```
class bank_actv_seq extends uvm_sequence#(mem_seq_item);

`uvm_object_utils(bank_actv_seq);

   function new(string name = "bank_actv_seq");
      super.new(name);
   endfunction: new

        virtual task body();
                `uvm_do_with(req,{req.c == 2;})
        endtask
endclass: bank_actv_seq

class bank_rd_seq extends uvm_sequence#(mem_seq_item);

`uvm_object_utils(bank_rd_seq);

        function new(string name = "bank_rd_seq");
                super.new(name);
        endfunction

        virtual task body();
                `uvm_do_with(req,{req.c == 4;})
        endtask
endclass: bank_rd_seq
```

```
class bank_pre extends uvm_sequence #(mem_seq_item);

`uvm_object_utils(bank_pre);

        function new(string name = "bank_pre");
                super.new(name);
        endfunction

        virtual task body();
                `uvm_do_with(req,{req.c == 5;})
        endtask
endclass

class ddr_rd_seq extends uvm_sequence#(mem_seq_item);

bank_actv_seq b_act;
bank_rd_seq b_rd;
bank_pre b_pre;

`uvm_object_utils(ddr_rd_seq);

        function new(string name = "ddr_rd_seq");
                super.new(name);
        endfunction

        virtual task body();
                `uvm_do(b_act);
                uvm_report_info(get_full_name(),"b_act",UVM_LOW);
                `uvm_do(b_rd);
                uvm_report_info(get_full_name(),"b_rd",UVM_LOW);
                `uvm_do(b_pre);
                uvm_report_info(get_full_name(),"b_pre",UVM_LOW);
        endtask
endclass
```

FIGURE 4. Sequence for read command

3.4 Monitor

A monitor is the passive element of verification surroundings. It just certainly tests the dut signal comes from interface yet it does not power them at all [2]. Monitor site visitor's collects coverage and send them to the numerous evaluation components like coverage and scoreboard. It really works equal as driver but without triggering dut signals [1].

3.5 Agent

Agent is essentially container which includes display, collection and driving force. It has active and passive mode of operation.

Active mode: In this driver and sequencer are instantiated and it drives the signals to the dut.

Passive mode: In this it will just sample the dut signal so only monitor is needed to instantiated.

3.7 Scoreboard

Scoreboard simply evaluate the reaction from dut against the expected reaction. So it delivers records about how typically reaction matched with favored / expected one [2].

```
forever begin
        wait(pkt_qu.size() > 0);
        mem_pkt = pkt_qu.pop_front();

        if(mem_pkt.wr_en) begin
                sc_mem[mem_pkt.addr] = mem_pkt.wdata;
                `uvm_info(get_type_name(),$sformatf("---::WRITE DATA::---"),UVM_LOW)
                `uvm_info(get_type_name(), $sformatf("Addr: %0h", mem_pkt.addr), UVM_LOW)
                `uvm_info(get_type_name(),$sformatf("Data: %0h", mem_pkt.wdata),UVM_LOW)
                `uvm_info(get_type_name(),"-------------",UVM_LOW)
        end
        else if(mem_pkt.rd_en) begin
                if(sc_mem[mem_pkt.addr] == mem_pkt.rdata) begin

                `uvm_info(get_type_name(),$sformatf("---::RAD DATA match::---"),UVM_LOW)
                `uvm_info(get_type_name(), $sformatf("Addr: %0h", mem_pkt.addr), UVM_LOW)
                `uvm_info(get_type_name(),$sformatf("Expected Data: %0h Actual Data: %0h",sc_mem[mem_pkt.addr], mem_pkt.rdata),UVM_LOW)
                `uvm_info(get_type_name(),"-------------",UVM_LOW)
                end
                else begin
                `uvm_info(get_type_name(),$sformatf("---::RAD DATA Mismatch::---"),UVM_LOW)
                `uvm_info(get_type_name(), $sformatf("Addr: %0h", mem_pkt.addr), UVM_LOW)
                `uvm_info(get_type_name(),$sformatf("Expected Data: %0h Actual Data: %0h",sc_mem[mem_pkt.addr], mem_pkt.rdata),UVM_LOW)
                `uvm_info(get_type_name(),"-------------",UVM_LOW)
                end
        end
end
endtask
```

FIGURE 5. Scoreboard Analysis

3.8 Environment

Environment is at top of structure in test bench and it could comprise more than one agents depending on the design/RTL. All the agents are connected in this layer.

3.9 Transaction Level Modelling

TLM, transaction-level modeling, is a modeling style for building highly abstract models of components and system. It relies on transactions, objects that contain arbitrary, protocol-specific data to abstractly represent lower-level activity.

Transaction Level Communication

Transaction-level interfaces define a set of methods that use transaction objects as arguments. A TLM port defines the set of methods to be used for a particular connection, while a TLM export supplies the implementation of those methods. Connecting a port to an export allows the implementation to be executed when the port method is called.

Transactions are passed as parameter to call the function, which might be blocking (suspend and wait for some event before returning) or non-blocking (return immediately) which is sufficient for basic synchronization within the verification environment.

The use of TLM interfaces isolates each component in a verification environment from the others. The environment instantiates a component and connects its ports/exports to its neighbor(s), independent of any further knowledge of the specific implementation. Smaller components may be grouped hierarchically to form larger components. Access to child components is achieved by making their interfaces visible at the parent level.

IV. RESULT

```
# UVM_INFO @ 0: uvm_test_top.env.mem_agnt.sequencer@@seq [uvm_test_top.env.mem_agnt.sequencer.seq] Start_of_body
#       ADDR = 3cb3     WDATA = c77501ed
#       ADDR = 1cff     WDATA = 2b62a34
#       ADDR = 3fe5     WDATA = e738df63
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 0: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
```

FIGURE 6. Data execution

```
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   39
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [uvm_test_top]     5
# [uvm_test_top.env]     6
# [uvm_test_top.env.mem_agnt]     8
# [uvm_test_top.env.mem_agnt.driver]     8
# [uvm_test_top.env.mem_agnt.monitor]     7
# [uvm_test_top.env.mem_agnt.sequencer.seq]     1
# ** Note: $finish    : /tools/mentor/questa10.2c-64/questasim/linux/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 0 ns  Iteration: 228  Instance: /tbench_top
[sumit.gorwadia@vnc3 ~/DDR2]$
```

FIGURE 7. UVM Report

Here in fig. 6, scoreboard shows what transaction is done on which memory location. The transactions made are all within the size limit. If there is an error in size overflow it will show an error.

V. CONCLUSION

One of the most challenging tasks was building the first UVM working test bench environment. The different components could not be tested individually, so all the environment had to be assembled in order to check efficiency of these components; making it hard to trace the source of errors. Hence the issues like this it was helpful to manage the way in how and which components reported debug messages. Automated test-cases are generated and given to the design. Functionality of The DDR2 memory protocol is been Verified. Developing the Verification IP for any design architecture it becomes actual modest by using UVM. UVM verifies the design in the most effective way.

REFERENCES

[1] Khaled Salah, "A UVM- Based Smart Functional Verification Platform: Concepts, Pros, Cons, and Opportunities", 9th International Design and Test Symposium, pp94-99, 2014, IEEE.

[2] B. Vaidya N. Pithadiya "An Introduction to Universal Verification Methodology" journal of information, knowledge and research in electronics and communication engineering, volume – 02, ISSUE – 02, 2013.

[3] S. Raghuvanshi, V. Singh "Review on Universal Verification Methodology (UVM) Concepts for Functional Verification" International Journal of Electrical, Electronics and Data Communication, ISSN: 2320-2084 Volume-2, Issue-3, March-2014.

[4] Young-Nam Yun, "Beyond UVM for practical SoC verification", SoC Design Conference (ISOCC), pp158-162, 2011.

[5] Universal Verification Methodology (UVM) 1.1 User's Guide, Accellera, 2011.

[6] On-line resources from http://www.doulos.com/knowhow/sysverilog/uvm/

[7] Online source from

https://www.scribd.com/doc/193965916/Uvm-Cookbook-Complete-Verification-Academy

[8] uvm-cookbook-systemverilog-guidelines-verification-academy-150116202822-conversion-gate02.pdf

[9] Juan Francesconi, J. Agustin Rodriguez, Pedro M. Juli´an, "UVM Based Testbench Architecture for Unit Verification", 2014 Argentine School of Micro-Nano electronics, Technology and Applications,

[10] C. Spear, System Verilog for Verification, Second Edition: A Guide to Learning the Testbench Language Features, 2nd ed. Springer Publishing Company, Incorporated, 2008.

[11] Janick Bergeron, Writing Testbenches: Functional Verification of HDL Models, Springer US, Kluwer Academic Publishers, 2003.

[12] Ashwin P. Patel, Vyom M. Bhankhariya, Jignesh S. Prajapati, "An Overview of Transaction-Level Modeling in Unviersal Verification Methodology", Journal of Information, Knowledge and Research In Electronics And Communication Engineering, 2013, IEEE.

[13] www.synopsys.com

[14] www.mentor.com

[15] www.design-reuse.com