Volume-6, Issue-2 (Sep-16)
ISSN (O) :- 2349-3585

Ethernet Verification using UVM Methodology

P. Arun Babu[1], K. Jaya Swaroop[2]
[2]Assistant Professor
[1,2] Department of Ece, ,
[1,2] Gandhiji Institute of Science and Technology, India

Abstract – Verification of Gigabit Ethernet MAC (Media Access Control) by using the most advanced verification methodology i.e. UVM (Universal Verification Methodology) has been presented in this paper. The main function of MAC is to forward Ethernet frames to PHY through XGMII (10 Gigabit Media Independent Interface) and to receive the frames from PHY to MAC through the same interface. Verification of IP provides an elegant way to verify MAC Characteristics such as frame transmission, frame reception etc. Coverage driven verification is best achieved by UVM with the use of factory and configuration mechanism, coverage metrics and self checking which reduces the time spent on verifying the design. A reusable testbench is developed by using UVM methodology which has been used to run different test scenarios on same TB environment. Regression testing of the design is carried out for achieving better coverage goal.

INTRODUCTION

In general, for verifying a SoC, firstly we need to verify the standard bus interconnecting IP Cores present in the system. The whole verification process of SoC consumes approximately 70% of total design time. In this research work, the problems taken care of are as follows:

1. Verification of Ethernet MAC which is an essential part of Ethernet SoC verification.
2. Development of VIP for MAC unit.
3.Using that MAC VIP, Ethernet MAC has been verified and coverage analysis has been performed.

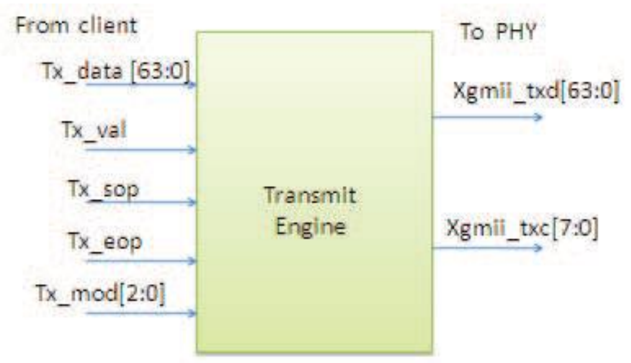
This VIP is a reusable verification component and henceforth it can be used to verify different SoC's designed using ETHERNET. The inbuilt tests employed in the ETHERNET VIP have given a jumpstart to achieve the required coverage goal which results in decrease of verification time. The UVM was introduced in December 2009 by Accellerawhich uses system Verilog as its base language. In recent years, Verification has become a very challenging task as more and more logic is being incorporated on a single chip. UVM Improves productivity and ensures re-usability. Maintenance of the verification components is much easier because the components are standardized.

PROPOSED SYSTEM

10Gigabit Ethernet MAC implements a MAC controller conforming to IEEE 802.3 specification. This proposed system consists of two modules namely transmit module and receive module. IEEE 802.3 data frame which consists of 7 different fields. These fields are set together to form a single data frame which illustrates the 7 fields: Preamble, Start-of-Frame delimiter, Destination Address, Source Address, Length, Data, and Frame Check Sequence.

A. Transmit Module

The transmit engine provides the interface between the client and physical layer. Fig. 1 shows a block diagram of the transmit engine with the interfaces to the client and physical



layer .

Fig.1. Block Diagram of ETHERNET Transmit Module

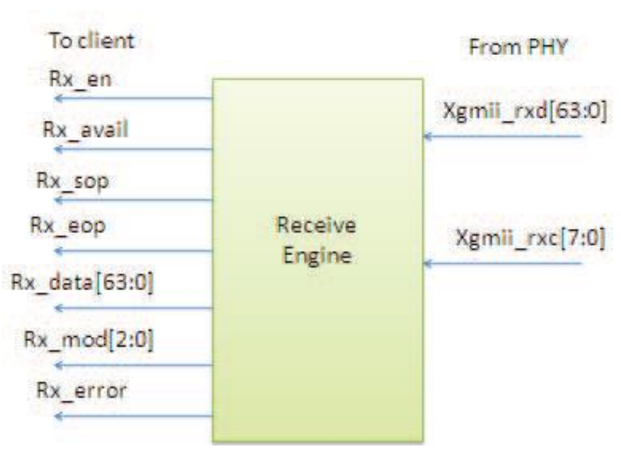


Fig.2. Block Diagram of ETHERNET Receive Module

B. Receive Module

The Receive Engine provides the interface between the physical layer and client. Figure 2 shows a block diagram of the receive engine with the interfaces to the client and physical layer. The following Fig. describes the components of ETHERNET MAC verification Architecture, which consists of verification components like agent, driver etc.

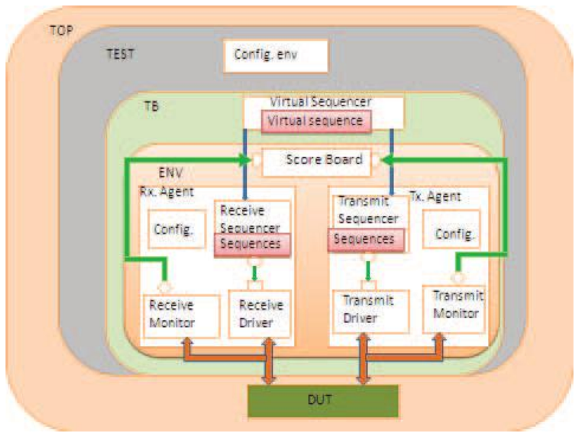


Fig.3. ETHERNET VIP Architecture

Data item represents the input to the device under Verification (DUV). Ethernet protocol specification gives clarity on valid attributes and values for Ethernet data packet. Generally data items are transmitted and generated to the DUV in a typical test. A large number of meaningful tests can be created by randomizing data item fields using SystemVerilog constraints thus maximizing coverage.

class transmit_xtn extends

uvm_sequence_item;

rand bit [7:0] data [7:0];

// add all the inputs and outputs

rand addr_t xtn_type;

rand bit[63:0] xtn_delay;

constraint a{mod inside{[0:7]};}

// add constraints for remaining

//variables

// standard uvm methods:

`uvm_object_utils_begin(transmit_xtn)

`uvm_unpack_intN(data, UVM_ALL_ON)

// include uvm methods for all the variables

`uvm_object_utils_end

Listing 1. Transaction class

A driver is an active component that mimics logic that drives the DUV. It Fetches data repeatedly from sequencer, drives the DUT based on the protocol using the virtual interface

class mac_tx_driver extends uvm_driver

#(transmit_xtn);

// connect phase method

vif = m_cfg.vif;

//-------run() phase method -------//

task mac_tx_driver::run_phase(uvm_phase

phase);

//----- task send_to_dut() method --//

task

mac_tx_driver::send_to_dut(transmit_xtn

xtn);

`uvm_info("MAC_TX_DRIVER",$sformatf("prin

tingfromdriver \n %s",

xtn.sprint()),UVM_LOW)

// Add the transmit logic

endtask

// UVM report_phase

Listing 2. Driver class

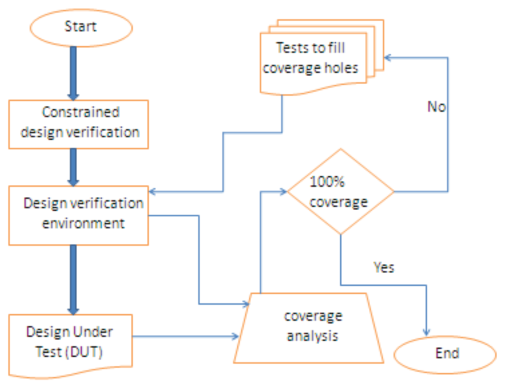

Start
Tests to fill coverage holes
Constrained design verification
No
100% coverage
Design verification environment
Yes
Design Under Test (DUT)
coverage analysis
End

Fig.4. Flow chart of MAC Coverage-Driven Verification

The items that are provided for execution to the driver is controlled by the advanced stimulus generator called sequencer. The sequences cannot directly access testbench resources, which are available in the component hierarchy. Using a sequencer, sequences can access testbench resources as a key into the component hierarchy. Upon the request from the driver, random data will be generated by the sequencer which controls the distribution of randomized values by allowing us to add constraints in the data item class .

```
class mac_tbase_seq extends uvm_sequence
#(transmit_xtn);
// Standard UVM Methods:
// Standard UVM Methods:
//------- task body method --------//
task mac_tx_xtns::body();
begin
req=transmit_xtn::type_id::create("
req");
strat_item(req);
// add asserts
assert(req.randomize() with {data[2]
inside {[20:0]}; en==1'b1; val==1'b1;
avail==1'b1;});
finish_item(req);
end
endtask
```

Listing 3. Sequence class

The monitor extracts signal information from the bus and translates it into transactions. Monitor is connected to other components via standard TLM interfaces like Analysis port and export.

To create a Monitor

1. Monitor class has been derived from the base class known as uvm_monitor
2. Added UVM infrastructure macros for class properties for the implementation of utilities for printing & copying,
3. Virtual interface has been declared in the monitor part for the connection between monitor and DUT
4. Obtained the data item from interface to send it to scoreboard.

```
class mac_tx_monitor extends uvm_monitor;
// Analysis TLM port to connect the monitor to the scoreboard
uvm_analysis_port #(transmit_xtn)
monitor_port;
// add Standard UVM Methods:
//------- build() phase method --//
super.build_phase(phase);
vif = m_cfg.vif;
//------ run() phase method -------//
task
mac_tx_monitor::run_phase(uvm_phase
phase);
forever
// Call collect data task
collect_data();
endtask
// Collect Reference Data from DUV IF
// add logic here
//Sending Receive transaction to SB
monitor_port.write(data_sent);
// UVM reort_phase
```

Listing 4. Monitor class

In an agent there are three specific components viz: sequencer, driver, and monitor. They can be reused independently. The driver, sequencer, and monitor are encapsulated by the agent. Verification environment can contain multiple number of agents. ETHERNET MAC has two agents: transmit and receive agent.

```
class mac_tx_agent extends uvm_agent;
//------- build() phase method ----//
super.build_phase(phase);
//------ connect() phase method --//
if(m_cfg.is_active==UVM_ACTIVE)
drvh.seq_item_port.connect(m_sequencer.se
q_item_export);
end
```

Listing 5. Agent class

A scoreboard is an analysis component that checks whether the DUT is behaving properly. Its function is to verify the proper action of the design at functional level by comparing the predicted output from reference model with the actual output from receiver .

```
class mac_scoreboard extends
uvm_scoreboard;
covergroup mac_fcov1;
option.per_instance=1;
cover_point_txmod: coverpoint
transmit_cov_data.mod
{option.auto_bin_max=8;}
// add cover points for all the data
//members
}
transmit_FC:cross txdata,txmod,tvalid;
endgroup:mac_fcov1
mac_fcov1=new();
//add Receive Operation - Functional
//Coverage
// Standard UVM Methods:
// ---- mac_transmit() method --------//
function void
mac_scoreboard::mac_transmit(transmit_xtn
tr);
if(txd_fifo_status ==0 && tr.full==1)
`uvm_info("MAC transmit function",
$psprintf("fifo_data=%b",fifo_data),
UVM_LOW)
begin
fifo_en=tr.val;
fifo_data = tr.data;
end
// add logic here
endfunction : mac_transmit
// add mac_receive() method
//--------run() phase ---------//
// explore the check_data
$cast(ref_xtn , re.clone());
if(mac_receive(ref_xtn))
begin
//compare
if(re.compare(ref_xtn))
begin
`uvm_info(get_type_name(),
$sformatf("Scoreboard - Data Match
successful"), UVM_MEDIUM)
xtns_compared++ ;
end
else
`uvm_error(get_type_name(), $sformatf("\n
Scoreboard Error [Data Mismatch]: \n
Received Transaction:\n %s \n Expected
Transaction: \n %s", re.sprint(),
ref_xtn.sprint()))
end
else
uvm_report_info(get_type_name(),
$psprintf("No Data transmitted in the data =%b \n %s",re.data,
re.sprint()));
```

Listing 6. Scoreboard class

Coverage Driven Verification of Ethernet. The environment acts as the top-level component for all the verification components. Environment, in addition to the agents, consists of scoreboard, coverage collectors and other analysis components.

```
class mac_env_config extends uvm_object;
bit has_functional_coverage = 0;
bit has_tagent_functional_coverage = 0;
bit has_scoreboard = 1;
bit has_tagent = 1;
bit has_ragent = 1;
//bit has_virtual_sequencer = 1;
mac_tx_agent_config m_tx_agent_cfg;
mac_rx_agent_config m_rx_agent_cfg;
int no_of_duts = 4;
// Standard UVM Methods:
```

Listing 7. Environment class

Interface is a static component that encapsulates communication between the hardware blocks. It provides a mechanism to group together multiple signals into a single unit that can be passed around the design hierarchy thus reducing the amount of code and promotes reuse. It is easy to maintain since signals can be added or removed easily. Interface ports work exactly like the module ports. When the

interface is instantiated, the connection of the port list can be done externally either by using order based or name based mapping . In the port list, only those signals are mentioned whose direction is same for both DUT and testbench like clock signal. But for the rest of the signals, instead of port list we need to specify the directions with the help of a modport.
Based on the declared directions, Modport restricts the interface access within a module. The function of the clocking block is to identify the clock signals and to capture the synchronization and timing requirements of the modeled blocks. Signals synchronous to a particular clock are assembled by a clocking block thus making their timing explicit and avoiding race around condition. With the help of the clocking block, testbench drives the signals on time. Interface can contain more than one clocking block depending on the environment. Set up and hold time of the DUV can also be modeled.

```
interface mac_if(input bit clock);
// transmiter driver CB
clocking tdr_cb @ (posedge clock);
default input #1 output #1;
output tx_data, tx_val,tx_sop,
tx_eop,tx_mod;
input tx_full;
endclocking
// add clocking block for the transmitmonitor , receive driver and receive monitor
//transmitter driver modport
modport tdr_mp(clocking tdr_cb);
// add mod ports for remaining components
Endinterface
```

Listing 8. Interface and Clocking block

Since Interface is static in nature and TB environment is dynamic, it is not possible to instantiate static interface in dynamic class objects. Virtual interface instance is created by using keyword “virtual”. By means of virtual interface drivers and monitors can be created and deleted dynamically during run time. For achieving coverage driven verification (CDV).UVM is always the best choice.

C. Test Cases

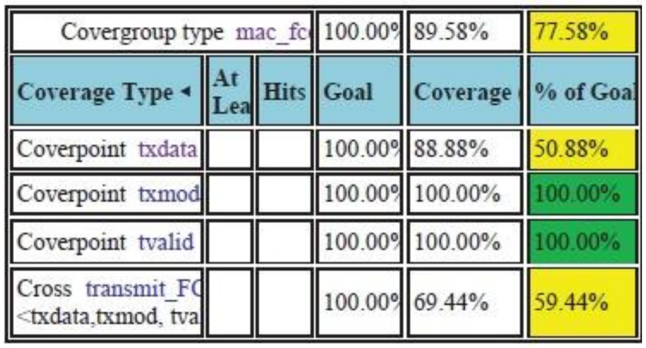
To check the functionality of the ETHERNET according to the specification the following test cases have been written:

Receive Enable
Receive available
Valid data (Tx and Rx)
Start of packet (Tx and Rx)
End of packet (Tx and Rx)
Modulus length (Tx and Rx)
Packets data(Tx and Rx)
Receive error
Transmit full

COVERAGE REPORTS AND RESULTS

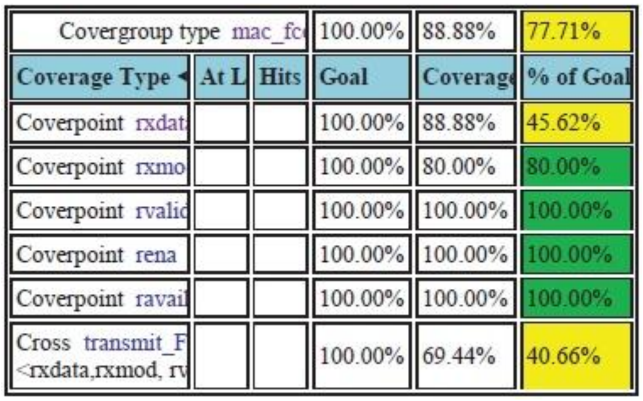
Verification results of Transmit Agent and the Receive Agent of the UVM Environment are presented above tables. According to Test Plan, the test cases are verified by developing the Verification IP for Ethernet Protocol. The Test Cases are written in the form of sequences in the Sequencer using System Verilog UVM methodology. The sequencer drives the sequences to the driver and thereby to Score Board. In the scoreboard, the actual output is compared with the expected one. If the obtained output matches with the expected result then we conclude that the verification is completed successfully. By using Questa simulation software, the Verification of Ethernet components such as transmit Agent and receive agent are done and the log files for the test cases are generated with Coverage report. Table II shows the coverage of the whole environment for code and functional coverage. 92.5% overall coverage has been obtained. It is not 100% as there is unreachable or unobservable code like testing logics, redundant code and functionality which is not under verification. Fig.5. shows Ethernet verification in uvm The Functional coverage has been attained by developing sufficient assertions and creating Cover groups, cover points and bins. 100% assertion coverage has been obtained the functional coverage and the code coverage of transmitter and receiver modules respectively. Tx coverage is 77.58% and Rx coverage is 77.71%. The cover group coverage is not 100% as all the registered address is not required to be checked which results in 89.23% coverage.

TABLE I. COVERAGE DETAILS FOR TRANSMIT PACKET COVER GROUP



Covergroup type mac_fc			100.00%	89.58%	77.58%
Coverage Type	At Lea	Hits	Goal	Coverage	% of Goal
Coverpoint txdata			100.00%	88.88%	50.88%
Coverpoint txmod			100.00%	100.00%	100.00%
Coverpoint tvalid			100.00%	100.00%	100.00%
Cross transmit_FC <txdata,txmod, tva			100.00%	69.44%	59.44%

TABLE II. COVERAGE DETAILS FOR RECEIVE PACKET COVER GROUP



Covergroup type mac_fc			100.00%	88.88%	77.71%
Coverage Type	At L	Hits	Goal	Coverage	% of Goal
Coverpoint rxdat			100.00%	88.88%	45.62%
Coverpoint rxmo			100.00%	80.00%	80.00%
Coverpoint rvalid			100.00%	100.00%	100.00%
Coverpoint rena			100.00%	100.00%	100.00%
Coverpoint ravail			100.00%	100.00%	100.00%
Cross transmit_F <rxdata,rxmod, rv			100.00%	69.44%	40.66%

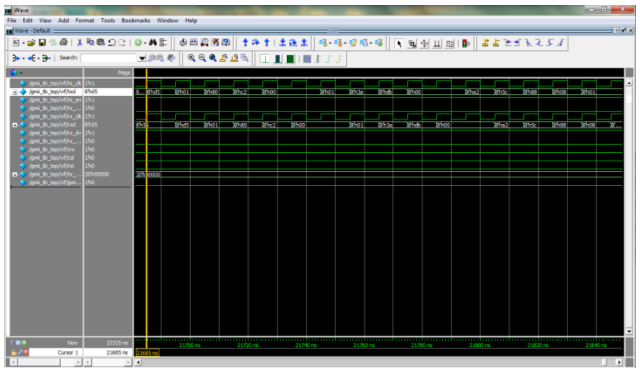


Fig.5. Ethernet verification in UVM

CONCLUSION

The specifications of MAC are verified successfully using UVM methodology on QuestaSim simulator. Functional coverage i.e. measure of implementation of design is carried out and 92.5% of coverage is extracted. The coverage can be improved by modifying the code according to the need. The scoreboard successfully compares the result of every transaction generated.

ACKNOWLEDGMENT

I express my sincere gratitude to Mr. K. Jaya Swaroop, Department of ECE , GIST college, India and also thanks to for continuous guidance and other Professors of Department of VLSI Design and Embedded Systems, GIST, India for extending their help & support in giving technical ideas about the paper without which I would not come up with this paper

REFERENCES

[1]. P.Chauhan, E.M. Clarke, Y.Lu and DongWang, “Verifying IPCore based System-On-Chip Designs”, Carnegie Mellon University Research Showcase.

[2]. Samanta, P, Chauhan, D, Deb, S, Gupta, P.K, “UVM based STBUS Verification IP for Verifying SOC Architectures”, Proc IEEE VLSI Design and Test, 18th International Symposium, doi. 10.1109/ISVDAT.2014.6881037, Coimbatore, July 2014.

[3]. Bhaumik Vaidya, Nayan Pithadiya, “An Introduction to Universal Verification Methodology”, Journal of information, knowledge & Research in Electronics and Communication Engineering, vol 2, Nov- 12 to Oct-13.

[4]. Assaf, M.H, Arima ; Das, S.R. ; Hernias, W, Petriu, E.M, “Verification of Ethernet IP Core MAC Design Using Deterministic Test Methodology”, IEEE International instrumentation and Mesurements TechnologyConference, doi.10.1109/IMTC.2008.4547312, victoria, May 2008.

[5]. Tonfat, J, Reis, R, “Design and Verification of a layer-2 Ethernet MAC classification Engine for a gGigabit Ethernet Switch”, Proc IEEE Electronics, Circuits, and Systems doi. 10.1109/ICECS.2010.5724475, Athens, Dec 2010.

[6]. Frazier,H. “The 802.3z gigabit Ethernet Standard”, Proc IEEE J, doi10.1109/65.690946, vol-12, May-June 1998.

[7]. MV Lau,, S. Shieh, Pei-Feng Wang, B. Smith, D. Lee, J. Chao, B. Shung, and Cheng-Chung Shih, "Gigabit ethernet switches using a shared buffer architecture,"Communications Magazine, IEEE, vol. 41, no. 12, pp. 76 - 84, dec. 2003.

[8]. Bergeron J, “Writing Test benches Using SystemVerilog”,Springer, ISBN-10: 0-387-29221-7, Business Media. 2006.

[9]. www.accellera.org/

[10]. www.testbench.in/ [online].