High Speed UVM Based Verification IP For Gigabit Ethernet Protocol

Selvakkani K
PG Scholar,
Department of VLSI Design,
Sri Shakthi Institute of Engineering and Technology
Tamilnadu, India.

Venkatesan K
Assistant Professor(S),
Department of VLSI Design,
Sri Shakthi Institute of Engineering and Technology
Tamilnadu, India.

Abstract

Increasing design complexity and concurrency of integrated circuits has made traditional directed test benches an unfeasible solution for testing. Verification may be a methodology used to demonstrate the functional correctness of a design. With automation human errors in process are minimized. Automation takes human intervention fully out of the Method. However, automation is not always possible, especially in processes that are not well defined process and continue to require human ingenuity and creativity, such as hardware design. Today testing is a word has been substituted with verification with increasing adoption of UVM, there is a growing demand for guidelines and best practices to ensure successful verification IP. In this paper, a complete truncation level verification environment based on UVM is proposed to tackle the verification barrier of complex gigabit Ethernet protocol IP. Ethernet has continued to be the most widely used network architecture today. The main aim of the project is verify the gigabit Ethernet IP with different interfaces. It also explains verification strategy and reuse of design environment with reference to verifying the Ethernet packet in Ethernet Intellectual Property (IP) Core. The verification atmosphere proposed in UVM will provide multiple levels of reuse, both within projects and between projects. The whole verification will done using system Verilog hardware description and Verification languages. We are using cadence Incisive simulator 12.2 for simulation.

Keywords— UVM, Verification IP, Gigabit Ethernet, TLM, MDIO.

1. INTRODUCTION

The improvement in the semiconductor manufacturing processes and the design methodologies led to the possibility of bigger and sophisticated digital styles. This scenario along with tighter time-to-market budgets reduces the possibility of success within the first attempt. To mitigate this example, industry drives to the development of new design methodologies such as the reuse of Intellectual Property (IP) cores design. In this context, the design verification acquires relevant attention from the academic and industrial environments. High confidence design verification means high quality designs and less circuit functional bugs transferred to the end user.
Design verification is the process to certify if the design functionality is according to the design specification. Two methods are commonly used:

Functional and Formal verification.

1. Functional verification or simulation-based verification is the most used technique for industrial applications. This technique is straightforward to cope with, however nearly always the foremost resource and "bottleneck" part of the look flow.
2. Formal verification methods were also proposed but only applied to low complexity circuits. In this scenario, few functional verification experiences of

communications systems were reported. Most of them were applied to processors designs.

Verification is a process used to demonstrate that the intent of a design is preserved in its implementation. Verification is always done in parallel to the design creation process that is verification is carried out at each step of manufacturing process. Verification is generally viewed as a fundamentally different activity from design. This split has led to the development of narrowly focused language for verification and to the bifurcation of engineers into two largely independent disciplines. This specialization has created substantial bottlenecks in terms of communication between the two groups. System Verilog addresses this issue with its capabilities for both camps. Neither team has to give up any capabilities it has to achieve success, but the unification of both syntax and semantics of design and verification tools improves communication. For example, whereas a design engineer may not be able to write an object-oriented test-bench environment, it is fairly straightforward to read such a test and understand that what is happening, enabling both the design and verification engineers to work together to identify and fix problems. Likewise, a designer understands the inner workings of his or her block, and is that the best person is to write assertions about it, but a verification engineer may have a broader view needed to create assertions between blocks.

Another advantage of including the design, test-bench, and assertion constructs in a single language is that the test-bench has easy access to all parts of the environment while not requiring the specialized APIs. The value of an HVL is its ability to create high-level, versatile tests, not its loop constructs or declaration style. System Verilog is based on the Verilog constructs for that the engineers have used for many years.

2. Universal Verification Methodology(UVM)

The UVM (Universal Verification Methodology) was introduced in December 2009, by a technical subcommittee of Accellera. UVM uses the Open Verification Methodology as its foundation. Accellera released a version UVM 1.0 EA on May 17, 2010. UVM Class Library provides the building blocks needed to quickly develop well-constructed and reusable verification components and test environments. It uses system Verilog as its language. All three of the simulation vendors (Synopsys, Cadence and Mentor) support UVM today which was not the case with other verification methodology. Today, more and more logic is being integrated on the single chip so verification of it is a very challenging task. More than 70 present of the time is spent on the verification of the chip. So it is a need of an hour to have an common verification methodology that provide the base classes and framework to construct robust and reusable verification environment. UVM provides that.

UVM environment used for the functional verification for the digital hardware, primarily using simulation. The hardware or system is to be verified would typically be described using Verilog, System Verilog, VHDL or SystemC at any appropriate abstraction level. This might be behavioral, register transfer level, or gate level. UVM is expressly simulation-oriented, however UVM can also be used alongside assertion-based verification, hardware acceleration or emulation. UVM test benches are complete verification environments composed of reusable verification elements, and used as part of an overarching methodology of constrained random, coverage driven verification. If you currently run RTL simulations in Verilog or VHDL, you will be able to consider UVM as replacing whatever framework and coding style you use for your test benches. But UVM test benches are more than the traditional HDL test benches, which could wiggle many pins on the design-under-test (DUT) and rely on the designer to inspect a waveform diagram to verify correct operation. UVM Improves productivity and ensures re-usability. Maintenance of the verification components will be much easier because the components are standardized.

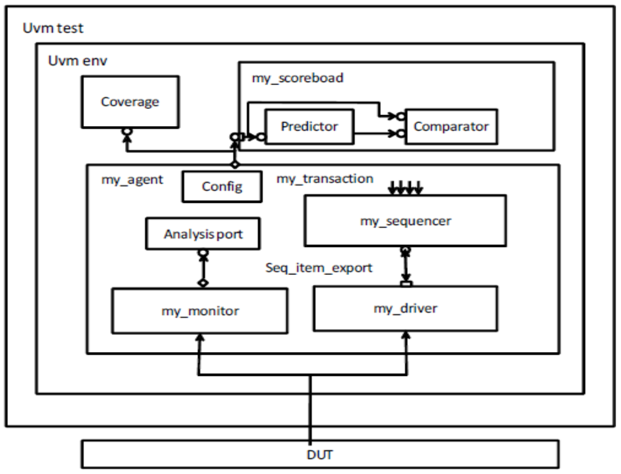

Figure 1: UVM architecture

2.1 GENERAL FEATURES OF UVM

2.1.1 Transaction-Level Modelling (TLM)

UVM uses TLM standard to describe communication between verification components in a UVM environment. One of the main advantages of using TLM is that in abstracting the pins and timing details. Dealing, the unit of data exchange between TLM components, encapsulates the abstract read of stimulus that can be expanded by a lower-level component.

2.1.2 Use of sequences for stimulus generation

The uvm_sequence and uvm_sequencer both provides the flexibility of running different streams of transactions without having to change the component instantiation. The transactions have to be compelled by an entity in the verification environment. Relying on a component to generate the transactions is limiting because it will require changing the component each time a different sequence of transactions is required. Instead UVM allows for flexibility by introducing uvm_sequence. uvm_sequence when started, register itself with a uvm_sequencer which is an uvm_component that acts as the holder of different sequences and can connect to other uvm_components.

2.1.3 Layering

Layering is a powerful concept in which every level takes care of the details at specific layers. UVM layering may be applied to components, which can be called hierarchy and composition, and to configuration and to stimulus. Typically there is a correspondence between layering of elements and objects. Layering stimulus, on the other hand, may reduce the complexity of stimulus generation.

2.1.4 Configurable

Configurable, an enabler to productivity and reuse, is a key element in UVM. In UVM, user can change the behaviour of an already instantiated component by three means: configuration API, Factory overrides and callbacks.

2.1.5 Re-usability

All the tenets mentioned above lead to another important goal which is reuse. Extensibility, configurability and layering facilitate reuse. Horizontal reuse refers to reusing Verification IPs (VIPs) across projects and vertical reuse describes the ability to use block-level VIPs in cluster and chip level verification environments.

3. PROPOSED SYSTEM

10Gigabit Ethernet MAC implements a MAC controller conforming to IEEE 802.3 specification. It is designed to use less than 2000 LCs/LEs to implement full function. It will use inferred RAMs and PADs to reduce technology dependence. A GUI configuration interface, created by test benches, is convenient for configuring optional modules, FIFO depth and verification parameters. Furthermore, a verification system was designed with test benches, by which the stimulus can be generated automatically and the output packets can be verified with CRC-32 checksum. In this proposed system consists of three module namely management module, transmit module, receive module.

3.1. Management module. This section describes the design of the management module for the 10-Gigabit Ethernet. MDIO function of this module part is designed to 10-Gigabit Ethernet IEEE 802.3 ae-2002. Configuration and statistic functions are also implemented. The MAC design is loosely based on the Xilinx LogiCORE 10-Gigabit Ethernet MAC, where the management function is similar with it too. The management module will be specifically designed to interface the client and the physical layer. The Management Module provides the manage interface to the client. Client can configure MAC and PHY via this interface.

Management Module is implemented with two sub-modules, which are:

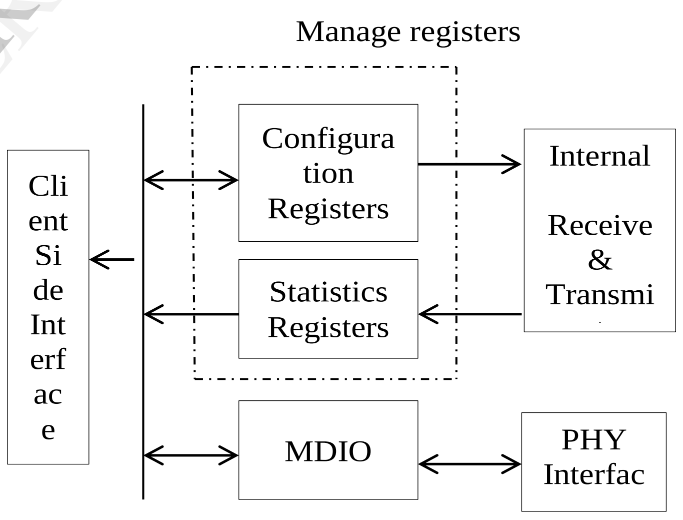

Figure 2: Internal Structure of the Management Module

This sub-module contains configuration and statistics registers. It responds read write requests from client side. Read write requests to MDIO registers will be dispatched to MDIO module. Figure 2 shows the internal structure of Management register sub-module.

3.2. Transmit module. The transmit engine provides the interface between the client and physical layer. Figure 3 shows a block diagram of the transmit engine with the interfaces to the client, physical, management and the flow control.

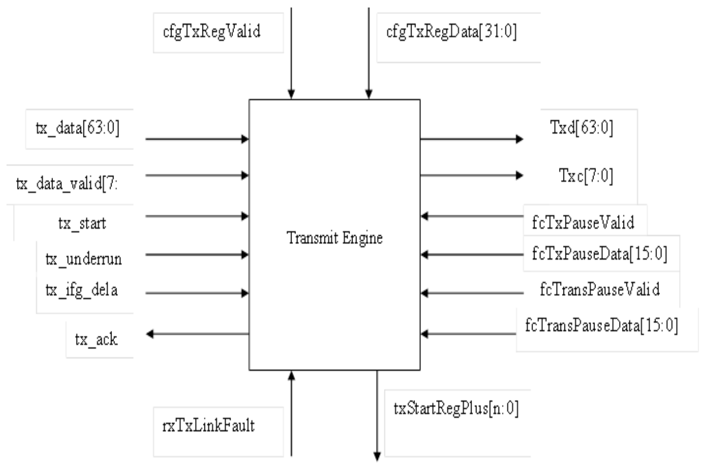

Figure 3: Block Diagram of the Transmit Block

The transmit engine contains several blocks; input and output FIFO/register, control logic and counters. The input and output FIFO/registers are employ to receive data from the client and distribute the data to the physical. All data flow is all under controlled from the control logic.

7	1	6	6	2	66 up to 1500			4
PRE	SDF	DA	SA	Type/ Length	LLC Data	------	PAD	FCS

Figure 4: IEEE 802.3x Frame

The tx_ack signal is generated using a type of counter circuitry to compensate when paused frame transmission is invoked by the flow control block or the inter frame delay is set at the start. The assertion of the signal is achieve when the count equal to the delay value. The request from the pause or inter frame will used to select the counter delay value. The minimum inter frame gap is 96 bits. For a normal transmission, the delay value will be 2 clock cycles.

The control logic is essentially a state machine that controls how the data is output to the physical by selecting between the control bytes and the client data. There are four different states in the control logic and there are IDLE, START, DATA and PAUSE.

1. In the IDLE state, IDLE bytes (07) are transmitted to the physical. When a receive fault occurs, the state machine will be stuck at IDLE until the signal is de-asserted.
2. In the START state, START control bytes, PREAMBLE bytes and Start Frame Delimiter are loaded into the output. Once the data is loaded, the state changed to DATA.
3. In the DATA state, the FIFO empty 64 bits at a time to the output until the empty flag is set. In this state, the tx_data_valid bytes are inverted and output to the command output, txc. If the empty flag is asserted in any of the FIFO, the output register with no valid data will be loaded with the TERMINATE and IDLE control bytes. After the last frame is transmitted, the state machine will either change to the IDLE state or when fcTransPauseVal signal is asserted, the state machine will change to the PAUSE state.
4. In the PAUSE state, a pause frame is sent out to the physical. In this state, the tx_ack is delayed until the pause frame is transmitted. The tx_ack signal is used to indicate to the client that the first data column is received.

7	1	6	6	2	2	2	42	4
P R E	S D F	D A	S A	Type/ Length	Op code	Pass time	Reser ved	F C S

Figure 5: PAUSE Frame

When the tx_underrun is asserted by the client-side, an error code will be inserted to the frame transmission. When VLAN is enabled, the transmitter is able to accept VLAN frames. Figure 6 shows the VLAN frames. The VLAN ID has a value of 8100 and the total frame size if extended to 1522 bytes.

7	1	6	6	2	2	2	46 up to 1500			4
P R E	S D F	D A	S A	VL AN ID	Tag cont rol	Type/ Lengt h	LLC Data	------	PA D	FCS

Figure 6: Virtual LAN Frame

When FCS is required for the data from the client, a parallel scheme is employed to generate the FCS. If the data is less than 46 bytes, padding is applied to the appropriate FIFO or registers. This is only achieved when the length of frame is received. The length will indicate if the frame is below the minimum frame size. By knowing the size, the appropriate FIFO or register can be applied with the padding. If the frame is greater than the maximum size, a counter is used to track the number of data columns and using the length field, this will determine when to truncate the data. An error code will be inserted to the transmission to corrupt the frame.

3.3 Receive Module: The Receive Engine provides the interface between the client and physical layer. Figure 7 shows a block diagram of the receive engine with the interfaces to the client, physical, management and also the flow control.
Receive Engine is enforced with nine sub-modules, which are:

3.3.1Receiver Clock Generator

This sub-module is used to generate internal reset and clock signals. In this design, DCM is used. Reset signal is generated from DCM locked signal.

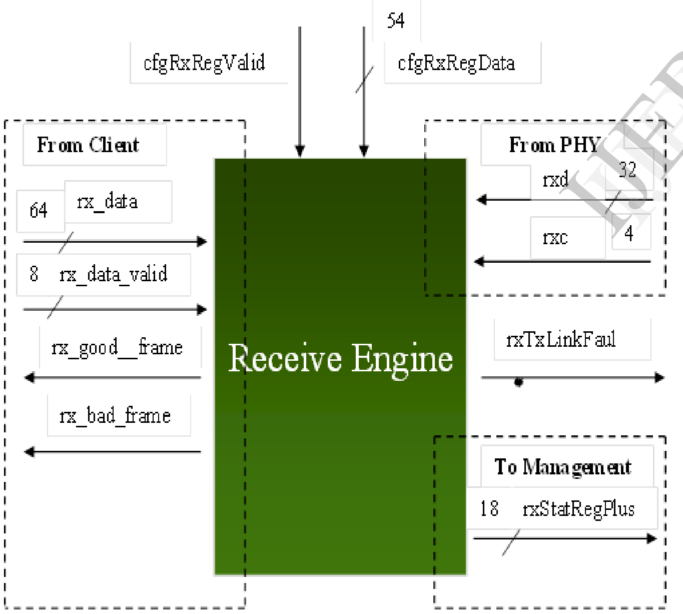

Figure 9: Block Diagram of the Receive Module

3.3.2 Receiver RS Layer

This sub-module implements the receive side function of Reconciliation Sub layer, which is define in IEEE 802.3ae Clause 46. This module samples xgmii_rxd and xgmii_rxc on both rising edge and falling edge of xgmii_rxclk. Besides, this module implements Link Fault Signaling.

3.3.3 Receiver Data Path

This sub-module is the main data path of Receive Engine. It has functions listed below:

1. Implements data pipeline;
2. Indicates SFD, EFD, and Error characters;
3. Gets Destination Address field and Length/Type field;
4. Indicates tagged and pause frame by checking Length/Type field;
5. Generates rx_good_frame and rx_bad_frame signals properly.
6. Manages Receive FIFOes, FIFOes are

a) Data FIFO: Data FIFO is used to store valid data from receiving frame. It is a 4K Bytes FIFO, which can store at least two frames.

b) Control FIFO: Control FIFO is used to store control signals, which is 512 Bytes. One bit in Control FIFO presents one byte in Data FIFO. If the bit is ‘1’, then its corresponding Byte is valid. If the bit is ‘0’, then the corresponding Byte is invalid.

3.3.4 Receiver CRC

This sub-module implements frame CRC checker. Its generating polynomial is:
G(x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2+x+1. It is composed of two CRC modules. One is 64bit input width, and the other one is 8bit, both are generated from easics. After checking the last byte of frame, if the magic sequence 32’hc704dd7b is generated, then the signal crc_check_valid is asserted; if the magic sequence is not generated, then the signal crc_check_invalid is asserted.

3.3.5 Receiver Destination Address Checker

This sub-module is used for check destination address of current frame. It checks four different types of destination address.

- ➢ Individual address: the MAC address of this MAC controller.
- ➢ Broadcast address: the broadcast address, whose destination address field is filled with all ‘1’s.
- ➢ Multicast address: this multicast address, which may be organized by user logic. This reversed MAC address is 01-80-C2-00-00-01. Control Frames will be sent with this destination address.

- Other address: the address of other stations. A frame which carries these destination addresses will be discarded. local_invalid signal is asserted when destination address lies in other address.

3.3.6 Receiver State Machine

This sub-module implements state machine of Receive Engine. There are six states:

- IDLE: Initial status. Controller starts receiving process when SFD received (get_sfd).
- rxReceiveDA: During this state, controller receives DA field.
- rxReceiveLT: During this state, controller receives Length/Type field.
- rxReceiveDATA: In this state, controller receives DATA field. Besides, it also watches DA invalid and Length invalid signals. Any invalid signals will turn state machine to rxGetError state. If no error happens in receiving states, controller can address rxIFGWait.
- rxGetError: During this state, controller stop receiving, dessert receiving signal (assert when controller is in rxReceiveDA, rxReceiveLT and rxReceiveDATA status) and return controller to IDLE state.
- rxIFGWait: It is sort of a turnaround state. It depends on the defined minimum gap between frames.

3.3.7 Receiver Number Counter

This sub-module is used for counting frame length. It simply may be a counter.

3.3.8 Receiver Length Type Checker

This sub-module is used for checking current frame's length. It takes three different situations into account: normal frame, tagged frame and jumbo frame.

- Normal Frame: Minimum Length: 64bytes; Most Length: 1518
- Tagged Frame: Minimum Length: 64bytes; Most Length: 1522
- Jumbo Frame: Minimum Length: 64byte; Most Length: 9K

3.3.9 Receiver State Module

This sub-module is used to collect statistic information of MAC controller.

4. SIMULATION RESULT

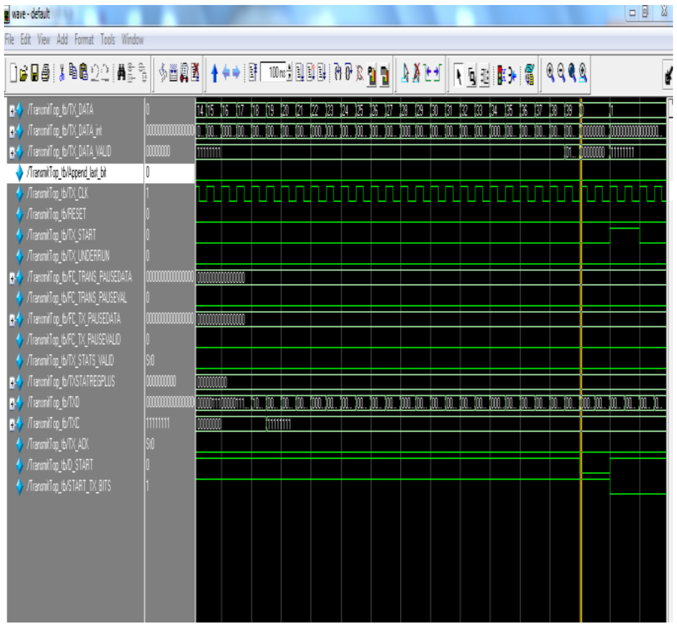
wave - default
File Edit View Add Format Tools Window
/TransmitTop_tb/TX_DATA
/TransmitTop_tb/TX_DATA_int
/TransmitTop_tb/TX_DATA_VALID
/TransmitTop_tb/Append_last_bit
/TransmitTop_tb/TX_CLK
/TransmitTop_tb/RESET
/TransmitTop_tb/TX_START
/TransmitTop_tb/TX_UNDERRUN
/TransmitTop_tb/FC_TRANS_PAUSEDATA
/TransmitTop_tb/FC_TRANS_PAUSEVAL
/TransmitTop_tb/FC_TX_PAUSEDATA
/TransmitTop_tb/FC_TX_PAUSEVALID
/TransmitTop_tb/TX_STATS_VALID
/TransmitTop_tb/TXSTATREGPLUS
/TransmitTop_tb/TXD
/TransmitTop_tb/TXC
/TransmitTop_tb/TX_ACK
/TransmitTop_tb/D_START
/TransmitTop_tb/START_TX_BITS

Figure 10: Frame Transmission with client supported FCS

In the above figure its shows a normal transmission from the client-side. FCS has to be generated if it is not included with the data from the client-side. If the data width is below 46 bytes, padding is needed to bring it in line with the minimum frame size. A parallel scheme has to be employ to generate the FCS.

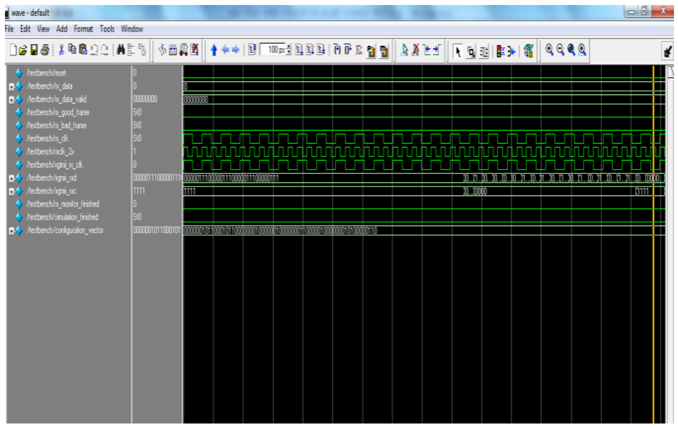

Figure 11: Frame received at Receiver side interface

5. CONCLUSION

In this paper, an outline of the Universal Verification Methodology was discussed. Verification is the most important part in designing the SoC and VIPs. Universal Verification Methodology (UVM) is one of the most famous Verification Methodology for verifying a complex IP and SoC. In this paper, Gigabit Ethernet is analyzed and its protocol is verified based on the normal testing methods using the test benches. The simulation result shows that the proposed method increases the time complexity and it has the test coverage up to 30%. In Normal verification method we cannot reuse the verification components for the projects, hence the UVM verification is used to improve the test coverage and its verification environment can be reused for the projects.

6. References

[1] Anjali Vishwanath, Ranga Kadambi, Infineon Technologies Asia Pacific Pte Ltd Singapore.

[2] Ben Chen, Cisco Systems, Shankar Hemmady, Rebecca Lipon of Synopsys, Verification IP reuse for complex networking ASICs- eetindia.com.

[3] Bergeron J, (2006)"Writing Test benches Using SystemVerilog", Springer, ISBN-10: 0-387-29221-7, Business Media.

[4] Brendan Mullane and Ciaran MacNamee, Circuits and System Research Centre (CSRC), University of Limerick, Limerick, Ireland.

[5] Cadence Designs Systems and Mentor Graphics Inc., (Sep 2008), "Open Verification Methodology User Guide" Version 2.0, available from http://www.ovmworld.org .

[6] Developing a Reusable IP Platform within a System-On-Chip Design Framework targeted towards an Academic R&D Environment www.design-reuse.com/

[7] Hannes Froehlich, Verisity Design, Increased Verification Productivity through extensive Reuse, Design and Reuse, Industry articles.

[8] Iman S, (May 2008) "Step-by-step functional verification with SystemVerilog and OVM", Hansen Brown Publishing,ISBN-10: 0-9816562-1-8.

[9] Radu M.E , Sexton S.M,(August 2008) "Integrating extensive functional verification into digital design education,", IEEE Trans. On Education, Vol. 51, No. 3.

[10] www.google.com

[11] www.opencores.org