Reusable Verification Environment for verification of Ethernet packet in Ethernet IP core, a verification strategy- an analysis

[1]L.Swarna Jyothi

[1] Professor
JSS Academy of Technical Education
Bangalore, Research Scholar
Dr.MGR University, Chennai,

[2] Harish R

[2] Verification engineer

[3]Dr.A.S.Manjunath

[3] CEO and MD,
Manvish eTech Pvt Ltd
Bangalore

Summary

Design reuse and verification reuse are important to satisfy time-to-market requirements. Designer must be able to reuse Intellectual Property in the design as golden model. Reuse of verification environment across different designs of the domain saves time to market further and improves total design verification quality. The Physical Layer is a fundamental layer upon which all higher level functions in a network are based. However, due to the plethora of available hardware technologies with widely varying characteristics, this is perhaps the most complex layer in the OSI architecture. The implementation of this layer is often termed Physical layer device (PHY). The Physical Layer defines the means of transmitting raw bits rather than logical data packets over a physical link connecting network nodes. A PHY chip is commonly found on Ethernet devices. Its purpose is digital access of the modulated link and interface to Ethernet Media Access Control (MAC) using media independent interface (MII) interface. This paper discusses Verification process, issues involved in verification process and Test Methodologies. A broad outline of the comparison of traditional verilog and specman verification methodologies has been presented here. It also explains verification strategy and reuse of design environment with reference to verifying the Ethernet packet in Ethernet Intellectual Property (IP) Core. Design Reuse is achieved through verilog tasks which were used in specman environment. Ethernet Phy e Verification component (eVC) is an in house development. Ethernet eVC is built with phy as a separate eVC and host being a task driven verilog Bus functional model (BFM). This allowed us to create a virtual host environment using a combination of verilog BFM and eVC. Verification environment reuse for different application with different interface is done by developing a wrapper around the Design Under Test (DUT) interface and then interfacing it to the environment. A detailed test plan is made for the complete and exhaustive test for Ethernet MAC Receiver. Coverage goals, coverage obtained and coverage analysis indicate efficiency of the verification methodology.

Key Words:

Ethernet eVC, BFM, DUT, MAC, Verification, Reuse, Test methods, MDIO, Coverage

1. Introduction

Verification is a methodology used to demonstrate the functional correctness of a design. With automation human errors in a process are minimized. Automation takes human intervention completely out of the process [1, 2, 3]. However, automation is not always possible, especially in processes that are not well defined and continue to require human ingenuity and creativity, such as hardware design.

Another possibility is error due to human intervention by reducing it to simple and foolproof steps. Human intervention is needed only to decide on the particular sequence or steps required to obtain the desired results. It is usually the last step toward complete automation of a process. However, just like automation, it requires a well-defined process with standard transformation steps. The verification process remains an art that, to this day, does not yield itself to well-defined steps.

Choosing the common origin and reconvergence points determines what is being verified. These origin and reconvergence points are often determined by the tools used to perform the verification. It is important to understand where these points lie to know which transformation is being verified.

The main purpose of functional verification [9] is to ensure that a design implements intended functionality. without functional verification, one must not trust that the transformation of a specification document into design and Register Transfer Logic (RTL) code was performed correctly, without misinterpretation of the specifications.

** **This research is funded by All India Council for Technical Education under Research Promotion Scheme**

Manuscript received November 5, 2008
Manuscript revised November 20, 2008

Figure 1 shows an overview of the steps involved in verification of a design under test.

Verification activity consists of driving vector stream into device and checking the vector stream coming out of the device. Higher level languages [12] are needed, as High-level language eases the creation of an expected value generator. Data check verifies the data correctness while temporal check verifies timing and protocol. Here the shared objects are used for input stimulus. It supports cycle-based behavior, events, and synchronizes with HDL simulator.

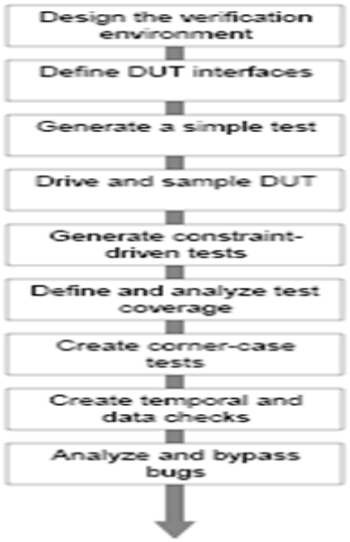

Figure 1. Generalized Verification Flow

Verification planning is an important and integral part of verification, irrespective of the size of the system. About 70% of the design cycle time is spent on verification; with proper verification planning some of the issues faced during the later phases of design can be addressed earlier. For SOC's it is observed that most of the peripherals are reused [7] from the previous design step with some modifications done on the feature set. Use of a pre-configured and pre-verified suite of code and IP means that adaptation and subsequent re-verification of the code for specific applications is greatly eased [4, 5, 8]. The reuse of verification code and methodology is a major factor providing significant reduction of the overall verification costs [6] One of the fundamental basics of design and verification reuse is the standardization of interfaces [10]. This has resulted in a number of interface and bus protocols that are used to connect different entities together. The logical conclusion for verification is to organize testbench components around those interfaces. Those components can then be used to verify multiple entities which have a particular interface. eVCs are verification components written in the e verification language [11]. The e language is designed specifically for verification. Reuse and extensibility are fundamental e language design principles.

The document is organized as follows: It discuss Verification process in Section 2, issues in verification in Section 3, issues in verification methodologies in section 4, Test methodologies in section 5, Specman based Verification in section 6 and Traditional verification based methodology in section 7. Section 8 presents a detailed case study with reference to a reusable Verification Environment for verifying Ethernet packet in Ethernet IP core. It discusses aspects such as, Management Data Input Output (MDIO), Verification strategy, Macro language, Test cases, Test Plan for Ethernet MAC receiver, Coverage Goals and the Bug file. Finally, Section 9 presents the conclusions drawn from the entire work.

2 Verification process

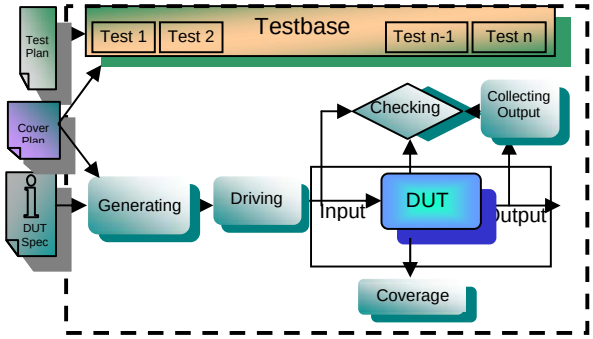

Figure 2. Generalized verification process

Figure 2 show clearly the components used in test environment like test plan, coverage to be achieved, generation of test vectors and its driving to the DUT

3. Issues in verification

Issues to be addressed during any verification activity are

- Capturing of all features of design (functional aspects).
- Compliance to all protocols.
- Coverage for all possible corner cases.
- Checking for locking states in the Finite State Machines (FSMs).
- Working of design in any random state.
- Elimination of redundant tests to save unnecessary simulation cycles and cost.
- Equivalence check with reference design.

- Methodology availability, for verification of each design modification.
- Generation of data patterns for directed or random test.
- Ease of generating flexible and modular test environment
- Reusability and readability of test environment.
- Handling of simulation time in automated test environment.

4. Issues in Verification Methodologies

Lack of effective automation during functional verification due to size and complexity of design, makes development of test environment, scheme for test generation and deterministic tests an intensive manual effort. Checking and debugging test results is also predominantly a manual process.

Design Complexity and size makes version control and tracking of design and verification process difficult, both at specification level and functional, which can often lead to architectural-level bugs that require enormous effort to debug. Debugging is always a problem, especially when they occur in unpredictable places. Even the most comprehensive functional test plan can completely miss bugs generated by obscure functional combinations or ambiguous spec interpretations. This is why so many bugs are found in emulation, or after first silicon is produced. Without the ability to make the specification itself executable, there is really no way to ensure comprehensive functional coverage for the entire design intent.

The relative inefficiency with which today's verification environments accommodate midstream specification changes also poses a serious problem. Since most verification environments are an ad hoc collection of HDL code, C code, a variety of legacy software, and newly acquired point tools, a single change in the design can force a ripple of required changes throughout the environment, eating up time and adding substantial risk.

Perhaps the most important problem faced by design and verification engineers is the lack of effective metrics to measure the progress of verification. Indirect metrics, such as toggle testing or code coverage, indicate if all the flip-flops are toggled or all lines of code were executed, but they do not give any indication of what functionality was verified. For example, they do not indicate if a processor executed all possible combinations of consecutive instructions. There is simply no correspondence between any of these metrics and coverage of the functional test plan. As a result, the verification engineer is never really sure whether a sufficient amount of verification has been performed.

5. Test Methodologies

Four prominent Test Methodologies are:

5.1 Deterministic

The oldest and most common test methodology used today is deterministic testing. These tests are developed manually and normally correspond directly to the functional test plan. Engineers often use deterministic tests to exercise corner cases, specific sequences that cause the device to enter extreme operational modes. These tests are normally checked manually. However, with some additional programming the designer can create self-checking deterministic tests.

Although deterministic testing offers the verification engineer precise control, providing accessibility to hard-to-reach corner cases, it has several drawbacks. Generating deterministic tests is a time-consuming, manual programming effort. Although simple tests can be written in minutes, the more complex ones can take days to write and debug. For example, to test a corner case requiring that two asynchronous data streams reach a specific point at exactly the same time, the verification engineer might have to resort to trial-and-error methods, running the test, seeing how far off it is, correcting it, and trying again. Moreover, midstream changes to the design is temporal behavior can cause the engineer to go through this process repeatedly. When this test is completed, the corner case is tested through only one possible path.

An average project normally develops many hundreds of deterministic tests, which is easily spread over several man-months to create. Checking deterministic tests also consumes considerable time and resources, whether it is performed manually or written into the test.

5.2 Pre-run generation

Pre-run generation is a newer methodology for generating tests that addresses some of the productivity problems associated with deterministic testing by automating the test generation process. C or C++ programs (and sometimes even VHDL and Verilog, despite the lack of good software constructs) are usually used to create the tests prior to simulation. The programs read in a parameter/directives file that controls the generation of the test. Often these files contain simple weighting systems to direct the random selection of inputs.

The generator normally outputs the test into a file, which is then read by the simulator and stored in memory. The simulator reads the next entry whenever it is prepared to

inject the next set of inputs. Although pre-run generation provides much higher throughput than deterministic testing, it is difficult to control. The parameters are static and do not provide much flexibility. Also, most generators of this type do not allow interdependencies between data streams if each data stream is generated independently. This can cause the generator to generate unlikely or even illegal tests.

Reaching corner cases using pre-run generation is nearly impossible. The engineer has very little control over the sequences generated. This makes it difficult to force the occurrence of specific combinations. As such, pre-run generation makes a suitable complement to deterministic testing, but cannot replace it.

Another problem with pre-run generation is that it is hard to maintain. As the verification process progresses, new parameters are often needed. This normally requires modifying the program, sometimes affecting delicate interdependencies between different parts of the generator.

Maintenance problems can also occur when updating the program after a bug is found in the RTL design. To temporarily avoid generating the same bug test sequence again, the engineer must modify the code until the bug is fixed in the RTL design. When the bug is fixed, the code must be again remodified. Several such modifications often coexist in the code. The modifying and remodiying process sometimes introduces bugs into the generator, which may not be noticed until several hours or days of simulation have transpired. The cost of developing and maintaining such a generator requires a minimum of several man-months per project, and increases significantly as the generation becomes more complex.

A side-effect of this methodology is that the full test is usually very large, since it is generated in advance. It is commonly loaded into a simulation memory at the beginning of the test and run from there. This significantly increases the memory requirements for simulation, often causing the simulator to swap memory. This can slow down the simulation by orders of magnitude.

5.3 Checking Strategies

The two most popular ways to determine test results are to compare them to a reference model or to create rule-based checks. Both of these checking methods must include both the temporal behavior and protocols of the device as well as the verification of data.

Reference models are most common for processor-like designs where the correct result can be predicted with relative ease. Designers usually develop the reference model in C or C++. Stimuli are injected into the reference model as well as the device, and their outputs are compared. In gray and white-box methodologies, the comparisons also include the state of internal registers and nodes.

Rule-based approaches are more common in communication devices for networking applications, where there can be several legal outputs for the same input, it is not easy to predict the correct result. In this case, the engineer often uses specialized techniques to check data integrity and protocols, such as scoreboarding, which tracks information about cells or packets without worrying about the order in which they appear on the output ports.

Engineers perform these checks either on-the-fly or post-run. Simple checks and protocol checks can be performed on-the-fly by the stubs and monitors using an HDL. Post-run checks are often performed using a C/C++ or PERL program. The outputs of the test are either saved in a simulator memory and then dumped into a file, or written into the file directly. The program reads the inputs , outputs and checks the correctness of the results. Often, these methodologies still require some amount of manual checking, usually achieved by viewing actual waveforms or data dumps.

The problem with these checking strategies stems from the way they are most commonly implemented today. Post-run checking wastes cycles. If a test runs for 500,000 cycles, but a bug occurred after cycle 2,000, then 498,000 cycles were wasted. In addition, since the post-run checking cannot detect a problem in real-time, the designer does not have access to the values of the registers and memories of the device at the time the problem occured. In general, debugging these problems requires rerunning the simulation to the appropriate point.

On-the-fly checking is more powerful. However, on-the-fly checks are most often implemented in Verilog or VHDL. These languages do not have a powerful temporal language to simplify protocol checks. They are low level and lack features like dynamic memory, which simplifies the process of writing the stubs/monitors and increases performance.

In addition, reference model checking is often hard to implement on-the-fly, since intermediate results are not always available. On-the-fly reference models also require a direct interface to the simulator (through PLI or FLI) which is not easy to write and maintain.

5.4 Coverage Metrics

Measuring progress is one of the most important tasks in

verification, and is the critical element that enables the designer to decide when to end the verification effort. Several methods are commonly used:

- Toggle testing verifies that over a series of tests, all nodes toggled at least once from 1 to 0 and back;
- Code coverage demonstrates that, over a series of tests, all the source lines were exercised. In many cases there is also an indication as to whether branches in conditional code were executed. Sometimes an indication of state-machine transitions is also available;
- Possibly the most common metric used to measure progress is to track how many bugs are found each week. After a period of a few weeks with very low or zero bugs found, the designer assumes that the verification process has reached a point of diminishing returns.

Unfortunately, none of the metrics described above has any direct relation to the functionality of the device, nor is there any correlation to common user applications. Neither toggle testing nor code coverage can indicate if all the types of cells in a communication chip with and without Cyclic Redundancy Check (CRC) errors have entered on all ports. These metrics cannot determine if all possible sequences of the instructions in a row were tested in a processor.

As a result, coverage is still measured mainly by the gut feeling of the verification manager, and eventually the decision to tape out is made by management without the support of concrete qualitative data. Not knowing the real state of the verification progress causes verification engineers to perform many more simulations than necessary, trading off CPU cycles for "confidence". This usually results in redundant tests that provide no additional coverage or assurance that verification is complete. The real risk is that the design will be sent to production with bugs in it, resulting in another round of silicon. The cost of re-spinning silicon includes non-recoverable engineering (NRE) costs to do the additional production process, the cost of extending the teams work on the project, and the major cost of reaching the market a few weeks late.

6. Specman based verification

Specman based verification is a methodology for functional verification that solves many of the problems of design and verification engineers encountered with today's methodologies. This is done by capturing the rules embodied in the specifications (design/ interface/ functional test plan) in an executable form. An effective application of this methodology provides four essential capabilities to help to break through the verification bottleneck.

- Automates the verification process, reducing the work needed to develop the verification environment and tests considerably.
- Increases product quality by focusing on verification effort to areas of new functional coverage and by enabling the discovery of bugs not anticipated in the functional test plan.

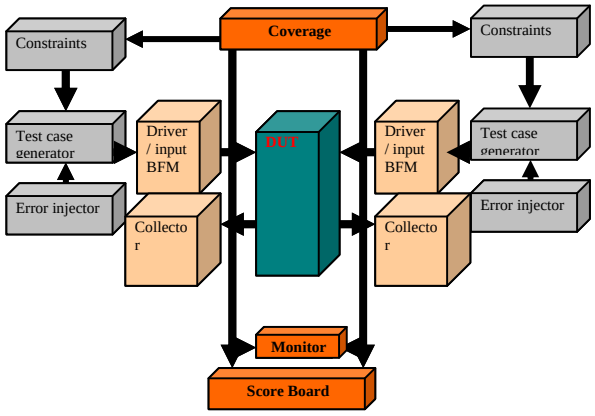

Figure 3. Generalized Verification environment

- Provides functional coverage analysis capabilities to help measure the progress and completeness of the verification effort.

Raises the level of abstraction used to describe the environment and tests from the RTL level to the specification level, capturing the rules defined in the specs in a declarative form and automatically ensuring conformance to these rules.

Figure 3 gives a generalized verification environment which can be used as guideline environment for developing verification environment for most of the designs. The environment consists of the components, Input BFM driver, Collector, Coverage, Test case generator, Error injector, constraints, Scoreboard and Monitor.

7. Traditional Verification Methodology

Some important points are:

7.1 Productivity & Quality Issues

Verification is more than 50% of an overall project cycle. It May require tens of thousands of lines of verification code. Design spec changes cause major verification delays. Implementing all identified tests in test plan within the project schedule is the Productivity issue.

7.2 Requirement for Productivity Improvement

The verification environment must be created and/or maintained efficiently. Human should spend more time at higher level details providing simulation goals, analyzing errors reported by checkers and providing more direction when goals are not being met.

7.3 Quality Issues

Verification complexity makes it a challenge to think of all possible failure scenarios. It does not provide a way to try scenarios beyond the expected failure scenarios.

7.4 Requirement for Quality Improvement

Confidence about the ratio of identified bugs must increase. An automatic way to know what has been tested must be available.

7.5 Task-based Strategy

To improve test-writing productivity higher level of abstraction is used for specifying the vector stream, and higher-level tasks are created in HDL or C. Task-based strategy limitations are, high test writing effort, many parameters values must be selected manually and high - level intent is not readily apparent.
There is no need to buy new tools or licenses and it provides homogeneous environment.

8. Case study:

A Verification Environment for verifying Ethernet packet in Ethernet IP core

When verifying the Ethernet IP core it is necessary to take several critical components into consideration. The interfaces to the host as well as the PHY, which posed extremely serious verification challenges. An automated test bench, creating a verification environment takes time. The time could be reduced by reusing common elements between designs and different applications developments.

In addition, as with many verification projects today, our goals were to develop a high-quality device in an extremely tight time schedule. This section describes our approach to the verification of this complex device and how we addressed the conflicting needs of quality versus complexity versus time.

The functional architecture consists of a host interface and a standard MII interface. The IP core consists of MDIO, Direct Memory Access (DMA) support, Configuration registers, Control logic, Transmitter FSM and Receiver FSM.

8.1 MDIO

The MDIO is a simple serial interface between the host and an external PHY device. It is used for configuration and status read of the physical device. A host processor responsible for system configuration and monitoring typically uses the MDIO to perform individual accesses to the various PHY devices.

It implements the IEEE 802.3 Clause 22 standard MDIO interface used in Ethernet systems up to 1Gbit/s. MDIO Master core allows access to registers within multiple connected Slaves. The features such as simple register based user application interface for the MDIO, MDIO frame generation with serial port tristate control, busy indication to user application during ongoing transaction are provided. PHY interrupt goes active when status change is indicated to application.

Host initiates an operation by writing into the configuration registers of the MAC. MDIO reads these registers and performs the tasks. It then reports it to the host by writing into the configuration registers which is polled by the Host continuously.

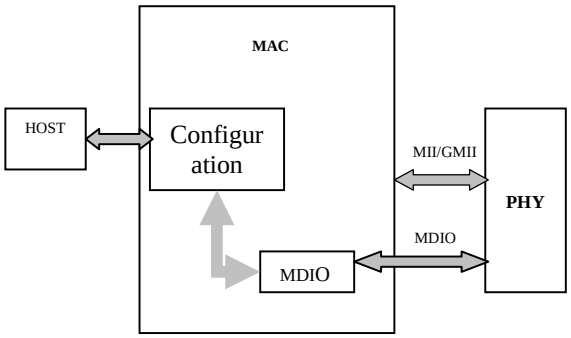

Figure 4. MDIO overview

A mux is used for selecting 8 bit field of MDIO frame at a time and loading it to parallel in serial out register. The PHY data is shifted out at the rising edge of PHY clock. The most significant bit of the data is shifted first. During a read cycle the data from PHY is shifted into the serial in parallel out register and a demux drives it to the required data bus.

The control block consists of counters which generates enable signal for the functioning of mux, demux, SIPO and PISO. It also generates the status signals to be written into the configuration registers. It also generates the PHY enable signal to drive PHY data as the data line is a shared tri-stateable bus, which is driven by the MAC for write transactions or by the PHY devices during read.

The clock generator module generates Management Data Clock by dividing host clock. The division factor is set in the configuration register field.

8.2 Verification strategy

Verification strategy for the Ethernet core was fairly sophisticated. The design was very complex and the verification team was tasked with ensuring as high a quality device as possible. Also, the verification team had the requirement that the environment lend itself easily to reuse for future generations, and that engineers who didn't create the environment be able to be productive within it as quickly as possible. The verification environment designed is shown figure 5.

Same components of the existing environment can be used for different application with different interface by developing a wrapper around the DUT interface and then interfacing it to the environment. Verification environment consists of BFM, test case generator, monitor and checker.

As with any commercial IP or SOC development with an aggressive timescale, the minimization of risk is key to delivery. For many commercial developers the decision to adopt a new verification paradigm can seem too risky.

The Specman Elite environment was built with the PHY eVC and the host eVC with few verilog tasks. This allowed us to create a virtual host environment using e masters, slaves and bus arbitrators.

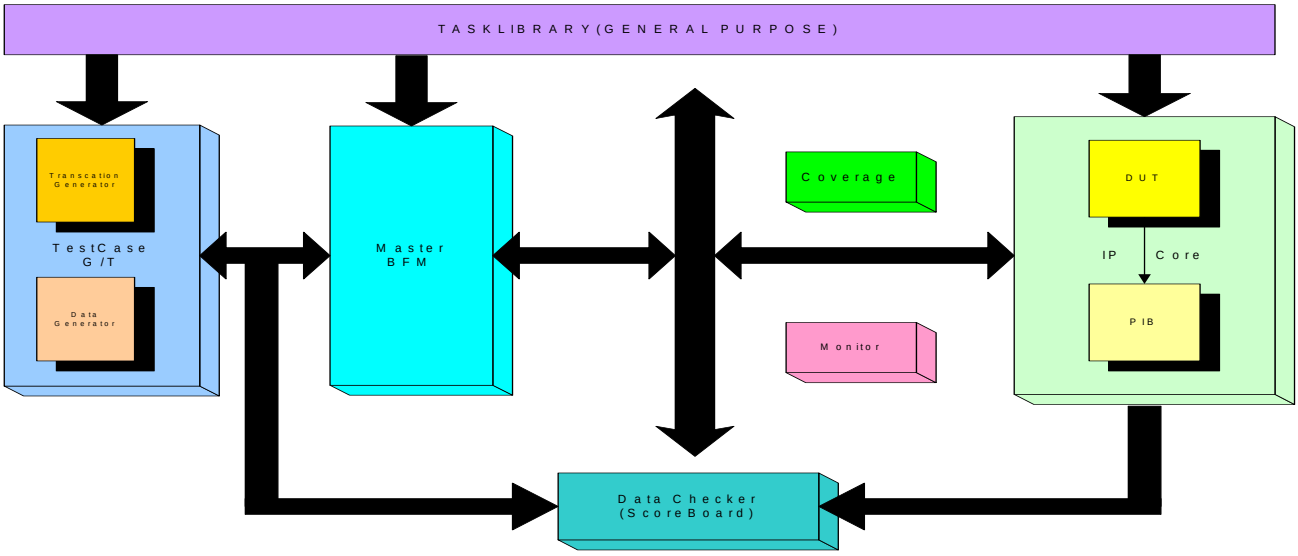

Figure 5. Verification environment

Specman Elite automated key aspects of the verification environment. Specifically, it works by generating stimuli into the device, inputs which can be fully random or fully directed. In this way, we can reach specific corner cases in the design without having to force a specific state. We can generate the inputs pre-run, on the fly, or a combination of the two.

. An eVC is a reusable piece of verification code written in e verification language with ERM (e reusable methodology) methodology. Specifically, the PHY eVC is a reusable verification environment developed in house. Just like an IP design core, this allowed time to be spent on verifying the unique aspects of the design rather than the standard components.

8.3. Macro language

One of the major objectives for the verification environment was the ability for the non-Specman "savvy" engineers to be able to easily use the environment to develop tests. We achieved this through “verilog tasks" that were built in a layered way using Specman Elite

macros. In fact, a huge number of the Ethernet verification tests are sequences of code similar to the above.

Through a constraint mechanism it was a simple matter to direct the traffic to be of say one burst length. Different frames and different configurations of the core were tested.

The complete verification suite in the Specman Elite environment comprised more than 56 test cases and randomization of the packet. This is an extremely small amount of tests given the complexity of the design, and it enabled us to get unparalleled coverage with a minimum number of tests and lines of code.

8.4 Test Cases

The test cases to check the functionality of the Ethernet Mac Receiver according to specification are:

1. Check the Reset condition of Receiver by making the RESET Bit 0 or 1 in RCB Block.
2. Check the Receiver Enable and Disable condition (RECEN=1)
3. Check the clock reception in case of Gigabit Media Access Interface (GMII)/MII.
4. Check the transfer of MII to GMII mode and vice versa.
5. Check for the GMII Mode.
    - Full Duplex Mode:
    - Half Duplex Mode
6. Check for MII Mode:
    - 10/100 Mbps (Full Duplex Mode)
    - Half Duplex Mode:

5 Check behavior of Receiver for Promiscuous Mode for the Different Frame reception.
6 Check the Successful Data reception.
7 Check for end of Reception (Receive status)

8.5. Test Plan for the Ethernet MAC Receiver

Various Environment components used in the verification of Receiver (Figure 6) are, Receiver DUT, BFM to drive different types of frames and various Inputs to the DUT, Monitor to monitor the various inputs and outputs of the Receiver DUT, Collector to collect the data from output of the DUT and Scoreboard to compare the data driven by the BFM and the output of the DUT. Figures 7, 8, 10 to 12 display the details of configuration for test plan. Figure 9 displays a detailed plan for verification of MAC receiver stages.

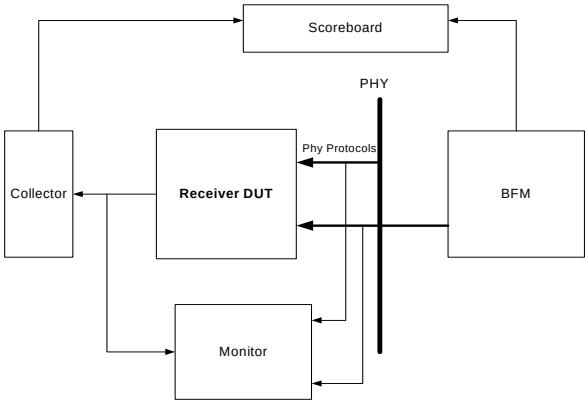

Figure 6. Receiver environment

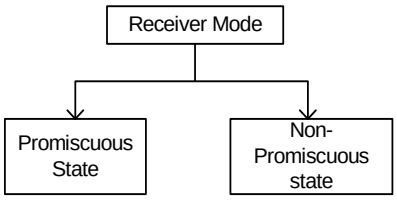

Figure 7. Modes of Receiver

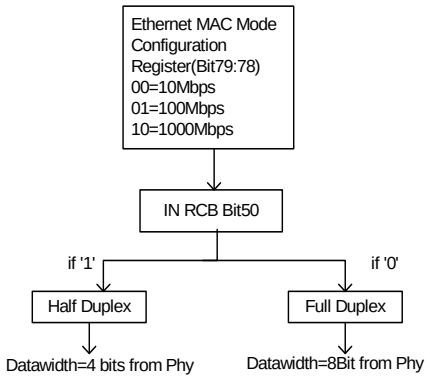

Figure 8.Receiver for modes MII/GMII at different speeds

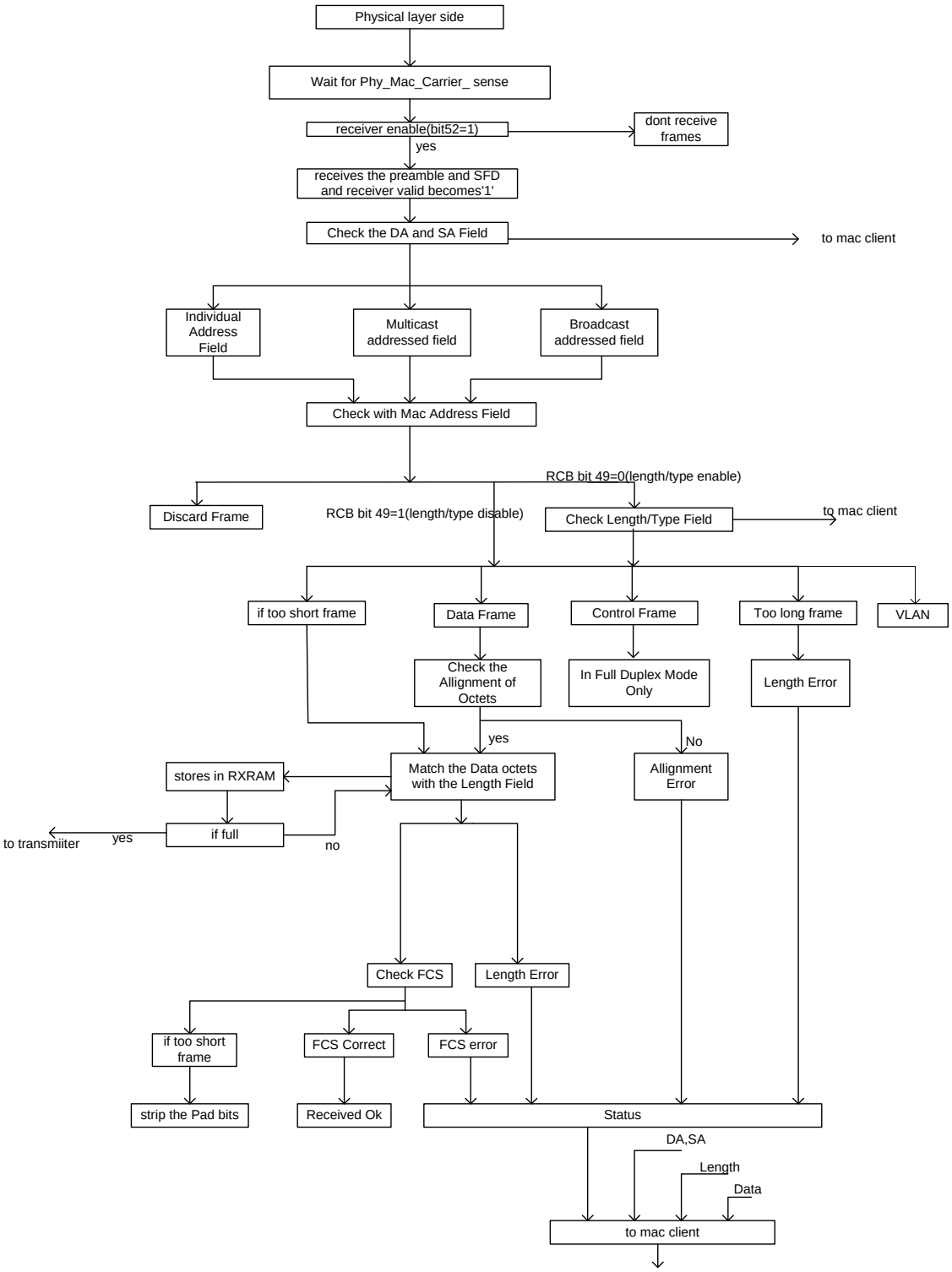

Figure 12. Verification of different stages of the receiver

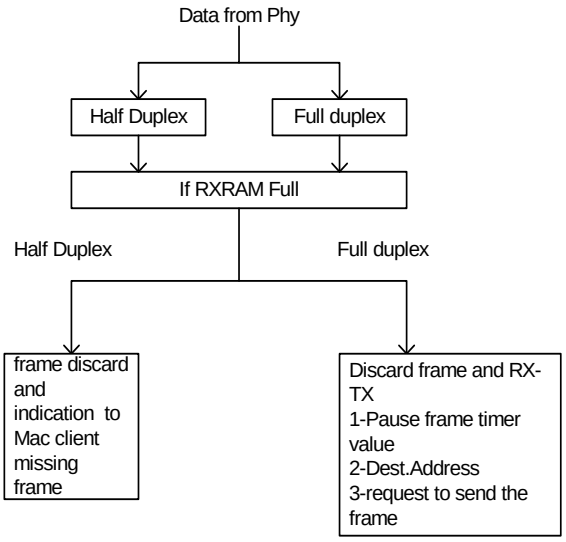

Figure 9. RXRAM Full condition at the time of reception of Frame for Half/Full Duplex mode

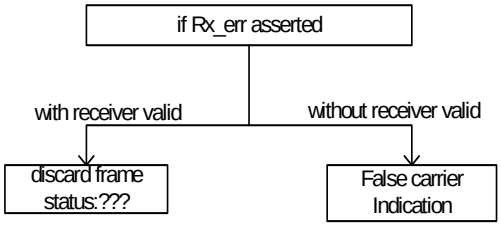

Figure 10. Collision Condition

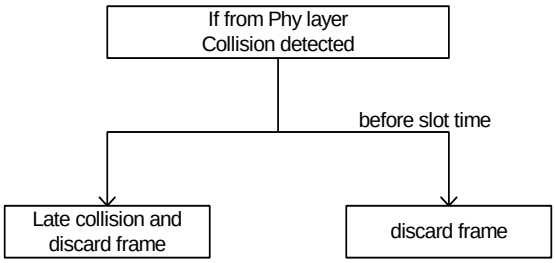

Figure 11. Rx Error from Phy Side

8.6 Coverage Goals and Details

Coverage goals
The line coverage to be achieved is 100%.
The condition coverage to be achieved is 85%.
The FSM coverage to be achieved is 100%.
The toggle coverage to be achieved is 90%.
The average coverage to be achieved is 90%.

Coverage tools/methodology followed
VCS Coverage Metrics (vcm) is used for coverage.

Coverage obtained
The line obtained is achieved is 100%.
The condition obtained is achieved is 90%.
The FSM obtained is achieved is 95%.
The toggle obtained is achieved is 90%.

Coverage Analysis
The average coverage achieved is 80%. Since Random testing is not done, the coverage achieved is not close to 100%. The offset is most in Condition Coverage and Toggle coverage. This is expected as not all combinations in a truth table will happen.

Team statistics : Team members – 2, No of man days – 82, Coverage > 90%

8.7 Bug File

==========
==========Bu
gNo:0 Block:RX Reported By:XXX Assigned
To:YYY Status:CLOSED
==========
==========Filed
on:10:09 8 Mar 2008 Closed on:06:49 9 Mar 2008
Resolved on: Title : Data frames greater than or equal to 494 are not being recieved properlyDescription : For data frames greater than or equal to 494 are not being recieved. The testcase is hanging. rx_ok is not coming for either of the frames.Logfile : Dumpfile : /export/home/sd27347/GBETHERNET/testbench/stand_alone/rx/simulation/run/rx_failing.shmFiles Modified :no files Comments :error in testbench and not in dut
==========
==========Bug
No:1 Block:RX Reported By:XXX Assigned
To:YYY Status:CLOSED
==========
==========Filed
on:02:38 9 Mar 2008 Closed on:09:00 9 Mar 2008

Resolved on: Title : data length greater than 1496 is reported as frame long error .same as for tagged frameDescription : Frame long error is coming for data frame length greater than 1496, it should only come for 1500 for untagged and for tagged it is 1504.Logfile : Dumpfile : Files Modified :fdb.v Comments :total cnt was checked for 1504 for untagged frames and 1508 for tagged frames and earlier we were checking for 1500 for both the cases
==========
==========Bug No:2 Block:rx Reported By:XXX Assigned To:YYY Status:CLOSED
==========
==========Filed on:03:42 17 Mar 2008 Closed on:05:09 20 Mar 2008 Resolved on: Title : receiver is having the delay to give rx_ok in case of short frameDescription :In case of Short frame reception, there is delay to get rx_ok, and this delay is depending upon the number of padding bytes in the frame. <enter detailed description of the bug here>Logfile : Dumpfile : Files Modified :fdb.v Comments :added a condition in fcs state "padded bytes are not counted with total cnt so min frame size 50 is taken"
==========
==========

9. CONCLUSION

Using a new verification methodology can appear daunting as it represents such a critical high-impact part of any IP development. The Ethernet project demonstrates that when facing a tough verification challenge, teams that take the challenge often find that the risks are in fact manageable and the benefits are significant. By using this approach , we were able to meet all our stringent requirements for the verification of this complex system. The extensibility of the e language, the macro facility and the power of Specman Elite's built-in generator were key elements that enabled this approach to be successful in short duration and smaller team effort compared to doing the same using complete verilog environment.

Acknowledgements

We thank Dr.S.Ravi, Professor and Head of ECE department, Dr.MGR Research and Educational Institute for the paper review. We also thank All India Council for Technical education for research funding and JSS Academy of Technical Education, Bangalore for providing the facilities. We Thank Ms. Chaya Asst. Professor, JSSATE for editing the script.

References

[1] Janick Bergeon ,Writing test benches – functional verification of HDL models
[2] Faisal I Haque. Et.al., The art of verification with VERA
[3] Samir Palnitkar, Design verification with e
[4] Brendan Mullane and Ciaran MacNamee, Circuits and System Research Centre (CSRC), University of Limerick, Limerick, Ireland, Developing a Reusable IP Platform within a System-on-Chip Design Framework targeted towards an Academic R&D Environment www.design-reuse.com/.
[5] Ben Chen, , Cisco Systems, Shankar Hemmady, Rebecca Lipon of Synopsys, Verification IP reuse for complex networking ASICs- eetindia.com
[6] 0. Petlin, *A.* Genusov, ASIC Alliance Corporation, L.Wakeman, Lucent Technologies, Methodology and Code Reuse in the verification of telecommunication SOCs, Proceedings of 13th Annual IEEE International ASIC/SOC Conference, 2000 (Cat. No.00TH8541) Verification Planning for Core based Designs
[7] Anjali Vishwanath, Ranga Kadambi, Infineon Technologies Asia Pacific Pte Ltd Singapore
[8] Kambiz Khalilian, Stephen Brain, Richard Tuck, Glenn Farrall, Infineon Technologies Reusable Verification Infrastructure for A Processor Platform to deliver fast SOC development**,** www.design-reuse.com/.
[9] Managing Functional Verification Projects Meeting the challenges of high-level verification in today's SoCs- Synopsys white paper-Oct 2007
[10] Hannes Froehlich, Verisity Design, Increased Verification Productivity through extensive Reuse, Design and Reuse, Industry articles.
[11] Verification Reuse Methodology, Essential Elements for Verification Productivity Gains- Verisity white papers-2002
[12] Accllera verilog LRM: http://www.accellera.org/home

Ms.L.Swarna jyothi is working currently as a Professor of Information Science and Engg Department at JSS Academy of Technical Education, Bangalore Visvesvaraya Technological University, INDIA. She has a total of 25years of academic experience. Her research interests are assertion-based verification, design for verification, verification reuse and VLSI Education. She has six reputed conference Publications in the related field. Her research has been funded by All India Council for Technical Education under Research Promotion scheme. She has already coordinated three projects sanctioned by AICTE and project grants are about 30 lakhs. She has received her Bachelors Degree in Engineering in Electronics and Communication Engg from Bangalore University and Masters Degree in Engineering in Comp Sc and Engg from Anna University, Chennai, INDIA. She is pursuing her PhD in Dr.MGR Research and Educational Institute, Chennai

Mr. Harish has 13 years of experience in reputed industries, out of which 8 years in ASIC/VLSI/Verification/Testing/Design Involving ASICs, Models, and SoCs. He has experience in test plan development, environment development, test cases implementation, models development, random testing and coverage. His Verification Experience include Hyper Transport Tunnel/Cave, Controller Area Network (CAN), PCIe, GB Ethernet, Flexray. He has expertism in Verilog, Specman e, ESEPro. VCS, Ncsim, Verilog-XL and Specview..His education includes Bachelor of engineering in Telecommunications and Master of Business Management from Bangalore University, Bangalore. Hereceived his MS (Software Systems) Degree from BITS, Pilani, India

Dr.A.S. Manjunath is the Managing Director and CEO of Manvish eTech Pvt, Ltd, Bangalore, INDIA. He has been a successful entrepreneur for over 12 years. He has done his PhD and UG degrees are from Bangalore University. His Masters Degree is form Mysore University India. He has worked as Professor and as an academician for over 13 years, in Bagalore University, India. He is guiding three PhD scholars and has 22 conference/journal papers to his credit.