uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Wael Hermas

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Approach to Coverage-driven Functional Verification of Complex Multimillion Gate ASICs

TITRE DE LA THÈSE / TITLE OF THESIS

S. Das

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. Nayak

T. Kwasniewski

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Approach to Coverage-Driven Functional Verification of Complex Multimillion Gate ASICs

Wael Hermas

A Thesis submitted to School of Graduate and
Postdoctoral Studies in partial fulfillment of the requirements for the
degree of Master of Applied Science, Electrical Engineering

April 2006

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-25783-8
Our file Notre référence
ISBN: 978-0-494-25783-8

To my family: my wife and my 3 boys

To my parents, brothers and sisters

Abstract

Today ASICs' designs are very complex and each consists of multi-millions gates. This creates a difficult and almost an impossible task for the verification engineers to verify thoroughly the whole design. The complexity of the verification effort grows exponentially. The most common verification methodology used today is Deterministic Testing. Based on today large designs, hundreds of deterministic test cases are required to verify the whole design. This is very time consuming and it requires lots of engineers' manpower. Since time to market is very essential, the engineers are forced to send the ASIC for fabrication even though lots of functionalities are not covered. A solution to this problem is Coverage-Driven Functional Verification (CDV). The CDV approach is based on ASICs functionalities. The verification process is done in the early stages of the design. In fact it is done right after the ASICs specifications are outlined and in parallel with RTL development. Functional Coverage is performed by collecting coverage items that correspond to ASICs functionalities. Using Functional coverage would minimize the number of test cases and enhance the verification process. In this thesis, the *"Ethernet IP Core"* Verilog design from OpenCores will be used. The Ethernet IP Core is a 10/100 Media Access Controller (MAC). It consists of a synthesizable Verilog RTL core that provides all features necessary to implement the Layer 2 protocol of the Ethernet standard. Both the Deterministic Testing Approach and the Coverage-Driven Functional Verification will be applied on the Ethernet IP Core design. Both verification methodologies will be compared and a conclusion will be driven to prove that CDV is the way for verifying today complex multi-millions gates ASICs. *Cadence Specman (e language* – IEEE 1647 e) is chosen as the verification tool because it provides different features for the Coverage-Driven Functional Verification Methodology.

Acknowledgements

I am really thankful to my supervisor Professor Sunil Das for his help, support, patience and invaluable guidance throughout the completion of my thesis. Also I would like to thank my co-supervisor Dr. Mansour Assaf for his help and guidance. He reviewed my work, implementation, structure of my thesis and provided me with constant feedback. I wouldn't be able to complete my thesis work without the help and the understanding of my wife. She was always supportive and there for me especially that my full-time engineering work and thesis requirements took lots of my time and left me with almost no time for my wife and three boys. Finally I would like to thank my parents, brothers and sisters for their support and belief in me.

Acronyms

eVC	:	e Verification Component
DVE	:	Design Verification Environment
eRM	:	e Reuse Methodology
Mac	:	Media Access Control
LXT971A	:	Intel Ethernet PHY Core
Wishbone	:	Host Standard Protocol
MII	:	Medium Independent Interface
CDV	:	Coverage-Driven Verification

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Problem Description

In recent years the complexity of large ASICs' designs has grown dramatically making design verification a huge bottleneck for large chip designs. In many companies, verification efforts consume most of the design resources and there are more verification engineers than designers, making verification the real limiter of time to market [3]. Also the verification effort has very high development cost. For example the verification effort for some of the large processors in IBM such as PowerPC Processor MAP1000 costs $3,000,000 [1]. The reason behind all these huge verification development costs is the verification methodology used in verifying complex large design. Basically hundreds or thousands of deterministic test cases are generated to try to cover all functionalities of the design. Unfortunately because of the time to market and lack of enough resources, companies are forced to send the design for fabrication even though they are not close to the completion stage of verifying all the functional characteristic of the design. Most likely only the major functions and corner cases are verified.

Maintaining thousands of test cases or importing them to system simulation would be impossible. So hi-tech companies are desperate to find a new and an efficient verification methodology to replace today's ones. After many researches in the field of ASIC Verification, it turned out that Coverage-Driven Functional Verification is the solution to enhance the verification process, minimize cost, and put products out to market on time without slipping any critical dates.

In this thesis, the problems and inefficiency of ASIC Verification is applied on a digital design from OpenCores "Ethernet IP Core: 10/100 Media Access Controller (MAC)" where deterministic testing approach is applied where all the test cases are performed through verilog tasks in one huge file consists of thousand lines. The verification of the OpenCores Etherent IP Core design is limited with directed test cases, lack of readability and maintenance capabilities.

1.2 Contribution

In this thesis, Coverage-Driven Functional Verification Approach was selected as the most efficient way to verify the Ethernet IP Mac Core design from OpenCores. Verification is accomplished based on designs' functional points. The functional points are driven early in the design cycles based on designs' specification. Using Specman Language, the most mature verification language in industry today, random generation / inputs are injected into the design. The functional items will cover the functions of the design and all the additional legal scenarios which the design should handle. Since random inputs are generated, all legal inputs will be injected on the design. Defining legal and illegal functional points, will verify the design not only in terms of legal actions, however it makes sure that illegal actions are not created by the design. So illegal functional points / items are defined and after running the random verification environment on the design, the illegal functional points are checked to make sure that the design wasn't in any time in these states. By focusing on legal and illegal design states, the design will be verified thouroughly and hence it will be free of bugs.

With CDV (Coverage-Driven Verification), late changes in the design would require small changes in the design verification environment. This can be easily done by updating the functional coverage points.
The concepts of Coverage-Driven Functional Verification are commonly used by large successful hi-tech companies such as Cisco Systems, Alcatel, Cadence and Intel. The industry shows that hundreds of large digital designs were taped out without bugs using CDV. In fact it was the only solution to make sure that designs are 100% free of bug.

1.3 Thesis Organization

This thesis is divided into 6 chapters. Chapter 1 provides an introduction of the thesis. Chapter 2 provides ASIC Verification Challenges and Solutions. Chapter 3 provides Ethernet IP Core (DUT) Verification Methodologies and Performance Measurement. Chapter 4 provides Ethernet IP Core Verificaton. Chapter 5 provides Ethernet Verification Environment Algorithms and Comparison. Chapter 6 provides a conclusion. Bibliography is provided at the end.

Chapter 2

2 ASIC Verification Challenges and Solutions

Advanced technologies such as design reuse and physical synthesis allow designers to create ever-larger designs. As gate counts exceed one million, verification methodologies have failed to adjust, turning verification task into the main bottleneck in the design process. Janick Bergeron wrote in his book "writing test-benches", "... the real problem is not how to create a 12 million gate IC that runs at 600MHz, but how to verify it" [9].

The magnitude and complexity of recent designs introduce new challenges. Two such major challenges are how to reduce the time it takes to do verification to a reasonable size, and how to ensure complete verification.

2.1 Effective Simulation

As exhaustive tests are practically impossible, the verification engineer should spend his time carefully. In order to avoid unnecessary repetitions, the engineer needs to be able to answer questions like: Were all possible stimuli variations injected? Were all possible results achieved? Were all Device Under Test (DUT) states visited? Did all the internal transitions take place? Did all the interesting events occur? By answering such questions we can re-set our simulation goals and dramatically expedite the verification process [48].

A Verification Engineer might think that he is covering all the design functionalities. Based on this concept he keeps running long regressions thinking that he is covering all the design functionalities. Most likely this is not true and lots of farm machines time is wasted, other verification engineers are constrained by lack of tools licenses and slipping schedules dates are definite.

2.2 Ensuring Completeness

The second challenge is to ensure completeness - how can we tell that verification is done? This decision is usually accompanied with tension and the emotional stress of trying to organize the abundant information gathered throughout the recent simulation sessions. Taping out an immature design can have an impact at both the personal and corporate levels. At the personal level, your job and professional reputation are at stake. It can be extremely embarrassing when a chip has to be re-spun due to a common use of the IC that was never verified. At the corporate level, small start-ups may literally collapse due to product quality, and even well established companies may find it hard to recover from the financial repercussions.

What about unnecessary delays in tape out? Tight schedules and narrow market windows characterize most of the industry today. Some companies quantify a one-day delay in the millions [48].

2.3 Verification Solution

Coverage-driven functional verification and Reuse Methodology is the solution for today ASIC verification challenges. The functional coverage is driven from RTL design specifications. Based on design specification, a test plan is written to cover all the functionalities of the design.

Much of functional verification is about knowledge representation. It is about taking the knowledge embedded in the various specifications (and in the implementation, and in people's heads), and representing it in a way that will be conducive to creating the four main components of verification: input generation, checking, coverage, and debugging. [38]

To achieve complete functional coverage without minimum and optimized a time a random input generator is developed and constrained in a manner where legal and interesting scenarios are produced to hit all the functional coverage points. The design verification environment should be developed in a way where checking and monitor are developed to check on the fly the integrity of the design. Based on functional coverage reports, the input generator should be constrained accordingly to narrow its generation to certain input streams which target in covering all the functional coverage holes.

ASIC verification is accomplished by developing a complete verification automation system. A successful complete verification automation system is the one which is built on the concepts of Coverage-driven functional verification and reuse methodology. The automation verification environment needs to have certain characteristics to be successful as written by Samir Palnitkar in his Design Verification With e book:

- A language that allows objects in the verification environment to be extended for a particular test rather than be derived by inheritance enables a 90% reduction in test writing labor.
- A language to express constrains in a verification environment because a constrained-based approach is more powerful for testbench description. A sophisticated constraint solver and generator are needed to solve constraints between items in different objects.
- A coverage engine that allows goals to be defined for complex test scenarios.
- A temporal engine that lets the engineer capture protocol rules in a concise, declarative syntax.

A complete verification automation system increased the overall productivity of a verification environment by helping the engineer efficiently perform the following tasks:

- Defining a test plan
- Writing and maintaining the testbench environment
- Selecting test vectors (input stimuli)
- Checking results
- Measuring progress against the test plan (coverage) [38]

The Specman Elite e language is the best and proven candidate fro a complete verification automation system. It consists with all the constructs and features necessary to build a complete verification automation system. The following diagram below shows how the Specman Verificaiton environment is built:

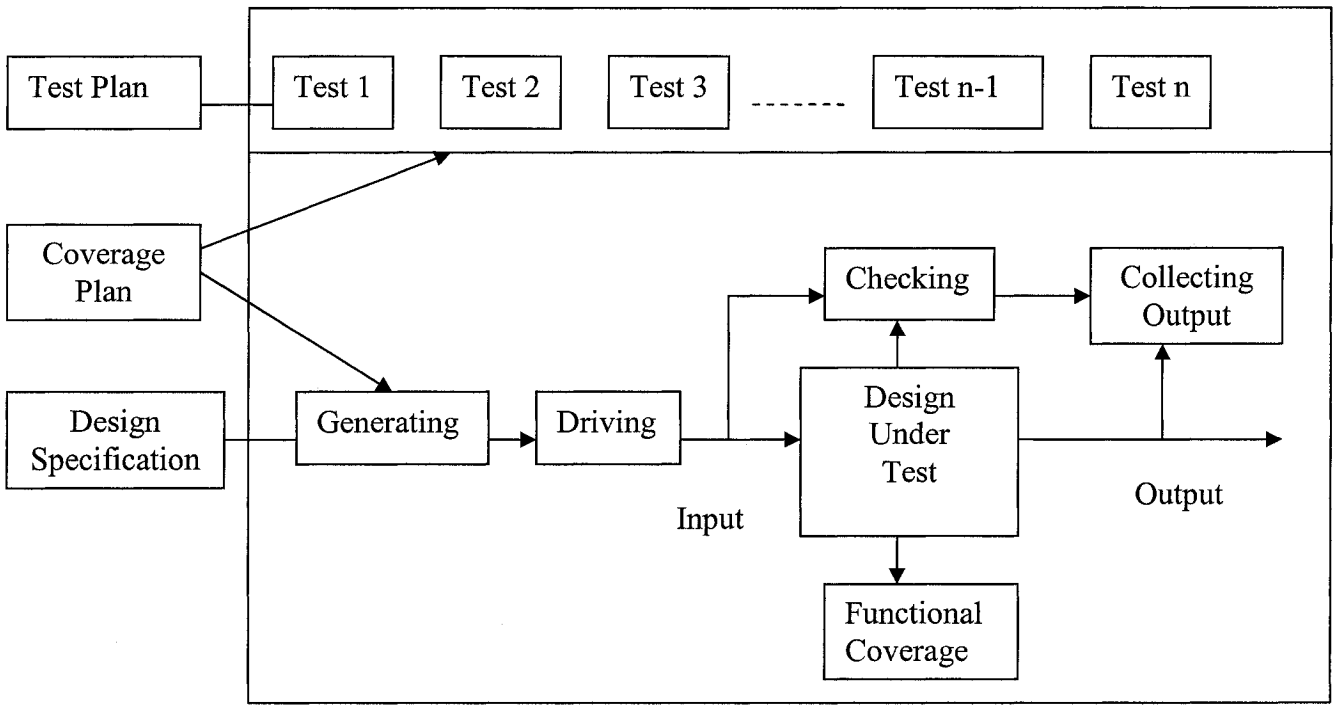

Figure 1: Verification Automation System Diagram

From the diagram we can see that to Specman Elite e language consists random input generator, checking / monitoring, functional coverage and debugging mechanism. From the design specification, a test plan is driven. The functional coverage points are outlined and an input generator is constrained to hit all the points in the functional coverage document. A checker / monitor is written to collect verify certain protocols, state machine transitions and data integrity. The checker looks at the RTL design interface

(signals) and collects interesting data. The Collection Output module takes both checker data and DUT and do a comparison for both data. If there is a mismatch error messages are produced and the simulation fails.

2.4 Related Work in Coverage-Driven Functional Verification

Scientists and engineers spend years trying to figure out a solution for today's verification problem. ASIC designs are getting more complex and verification tasks are getting exponentially more complex and hard to do. Most of the resources in any design development cycle are spent on the verification stage. So a solution must be found to do verification more efficiently and thoroughly.

Coverage-Driven Functional Verification is a methodology, which recently was adapted and considered as the ultimate solution for verifying multi-million gates ASICs. Previous studies and work were done on Coverage-Driven Functional Verification. In a research done at University of California, Irvine, coverage-driven functional verification was performed on pipelined architecture microprocessors. In this research, it was proposed a functional fault model that is used to define the functional coverage for pipelined architectures.

"To define a useful functional coverage metric, we need to define a fault model of the design that is described at the functional level and independent of the implementation details. In this report, we present a functional fault model for pipelined processors. The fault model should be applicable to the wide varieties of today's microprocessors from various architectural domains (such as RISC, DSP, VLIW and Superscalar) that differ widely in terms of their structure (organization) and behavior (instruction-set)."[39]

For example in this research / case study, the functional fault model was applied on Read/Write registers, operation execution, and pipelined execution.

"To ensure fault-free execution, all registers should be written and read correctly. In the presence of a fault, reading of a register will not return the previously written value. The fault could be due to an error in reading, register decoding, register storage, or prior writing. The outcome is an unexpected value. If *VRi* is written in register *Ri* and read back, the output should be*VRi* in fault-free case. In the presence of a fault, output *6= VRi* .

All operations must execute correctly if there are no faults. In the presence of a fault, the output of the computation is different from the expected output. The fault could be due to an error in operation decoding, control generation or final computation. Erroneous operation decoding might return an incorrect opcode. This can happen if incorrect bits are decoded for the opcode.

An implementation of a pipeline is faulty if it produces incorrect result due to execution of multiple operations in the pipeline. The fault could be due to incorrect implementation of the pipeline controller. The faulty controller might have erroneous hazard detection, incorrect stalling, erroneous flushing, or wrong exception handling schemes.

We define functional coverage based on the fault models. Consider the following cases:
- a fault in *register read/write* is covered if the register is written first and read later.
- a fault in *operation execution* is covered if the operation is performed, and the result of the computation is read.
- a fault in *execution path* is covered if the execution path is activated, and the result of the computation is read.
- a fault in *pipeline execution* is covered if the fault is activated due to execution of multiple operations in the pipeline, and the result of the computation is read."[39]
"In this experiment, the Specman Language was used. We developed our test generation and coverage analysis framework using Verisity's Specman Elite. We captured executable specification of the architectures using Verisity's "e" language. This includes description of 91 instructions for the DLX, and 106 instructions for the SPARC V8 architecture. We refer to these as *specifications*. We implemented a VLIW version of the DLX architecture using Verisity's "e" language."[39]

In this case study, we can see that Specman Language was used to facilitate the test generation and coverage collection. In this experiment, a functional fault model was developed to prove that is more sufficient than the normal directed and random directed test cases to verify complex designs such as microprocessor designs. In our thesis we use the coverage-driven functional verification as the proper and most efficient way to verify and produce free of bugs designs rather than the deterministic testing approach used by Open Core. In this thesis, coverage-driven functional verification methodology is applied on both legal and illegal functions of the design. Legal and Illegal functional points / items were defined and the design is considered free of bugs only when there is a 100% functional coverage. The illegal functional points are checked to make sure that the design was never in these illegal states. Otherwise there are faults in the design.

In the next few chapters, both the coverage-driven functional verification and the deterministic testing approach will be applied, compared and concluded for the Ethernet IP Mac Core from OpenCores. The Specman language will be used to generate the directed-random and random test scenarios and coverage reports will be produced and based on the coverage points' collections, directed test cases are written to get a 100% functional coverage. Only at this stage, we can consider that the complex design is free of bugs.

Chapter 3

3 Ethernet IP Core (DUT), Verification Methologies and Performance Measurement

Based on the Ethernet IP Core Product Brief document, the followings are the related information regarding the Ethernet IP Core [33].

3.1 Ethernet IP Core (DUT)

3.1.1 Introduction

The Ethernet IP Core is a 10/100 Media Access Controller (MAC). It consists of a synthesizable Verilog RTL core that provides all features necessary to implement the Layer 2 protocol of the Ethernet standard. It is designed to run according to the IEEE 802.3 and 802.3u specifications that define the 10 Mbps and 100 Mbps Ethernet standards, respectively.

3.1.2 Features

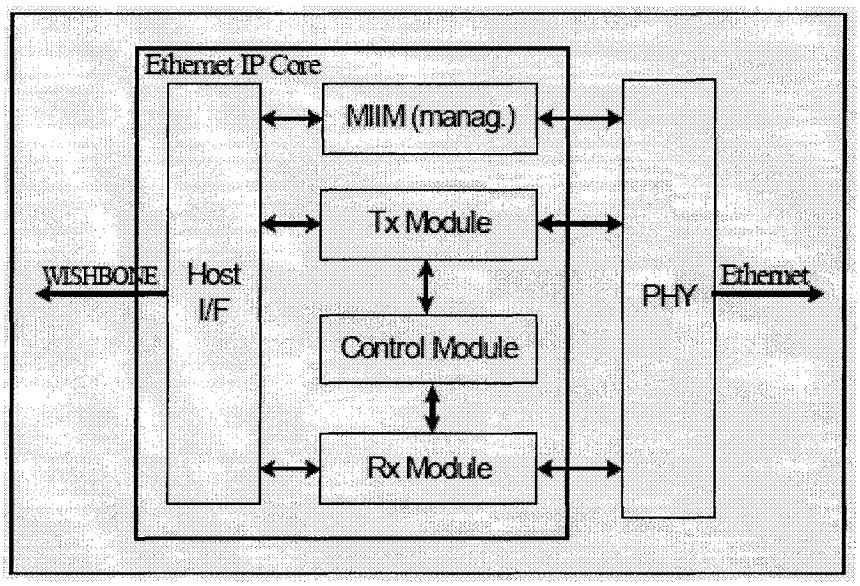

Figure 2: Core Architecture Overview

The core provides the following features:

• Flow control and automatic generation of control frames in full duplex mode (IEEE 802.3x).

• Collision detection and auto retransmission on collisions in half duplex mode (CSMA/CD protocol).
• Automatic 32-bit CRC generation and checking.
• Preamble generation and removal.
• Complete status for TX/RX packets.
• IEEE 802.3 Media Independent Interface (MII).
• WISHBONE SoC Interconnection Rev. B compliant interface.

3.1.3 General Description

3.1.3.1 Architecture

Adjoining figure shows the general architecture of the Ethernet IP core. It consists of several building blocks:
• TX Ethernet MAC (transmit function) block with the CRC generator
• RX Ethernet MAC (receive function) block with the CRC generator
• MAC control block
• Management block (MIIM)
• Host interface

3.1.3.1.1 TX and RX Modules

The TX and RX modules provide full transmit and receive functionality. CRC generators are incorporated in both modules for error detection purposes. The modules also handle preamble generation and removal. Padding occurs automatically (when enabled) in compliance with the IEEE 802.3 standard. When enabled, packets greater than the standard can be transmitted.

3.1.3.1.2 Control Module

The control module provides full duplex flow control, according to the IEEE 802.3u standard. Flow control is achieved by transferring the PAUSE control frames between the communicating stations.

3.1.3.1.3 Management Module (MIIM)

The management module provides the standard IEEE 802.3 Media Independent Interface (MII) that defines the connection between the PHY and link layers. Using this interface, the device (RISC) connected to the host interface can force PHY to run at 10 Mbps versus 100 Mbps to configure it to run at full versus half duplex mode.

3.1.3.1.4 WISHBONE Interface

The WISHBONE interface connects the Ethernet core to the RISC and to external memory. The core is WISHBONE SoC Interconnection specification Rev. B compliant. The implementation realizes a 32-bit bus width and does not support other bus widths.

3.1.4 Utilization

Technology	Area	Silicon Area	Speed	Power Consumption
Xilinx Virtex	2600 slices	100000 gates	60 MHz (25 needed)	TBD

3.2 Verification Methodologies

3.2.1 Traditional Deterministic Testing Approach

Deterministic Testing Approach mainly focuses on implementing the Functional Test Plan to verify a specific ASIC through writing hundreds of test cases. Each test plan item is implemented in a test case and run against the DUT. Special checking mechanism in the Design Verification Environment is embedded to verify that the DUT is behaving as expected. Once no errors are flagged, the test case is considered as a PASS. The following diagram and flow chart shows what mentioned above:

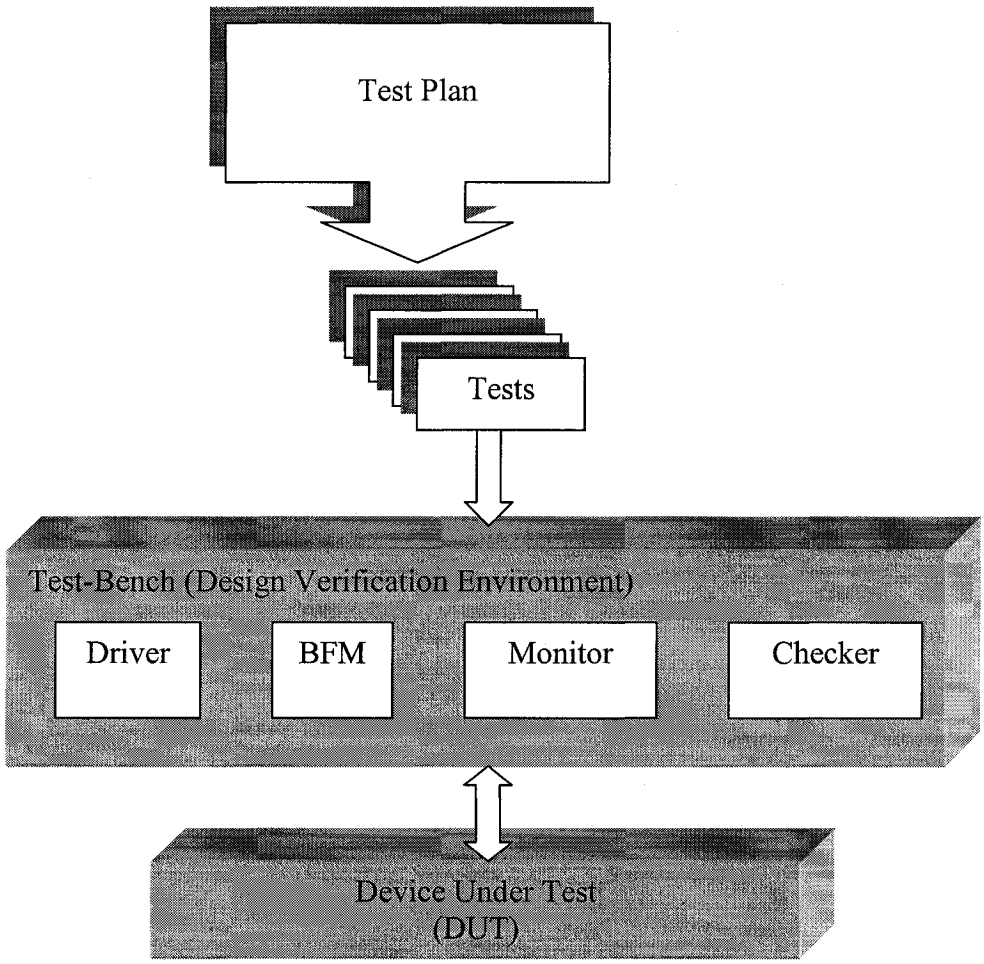

Figure 3: Deterministic Testing Approach

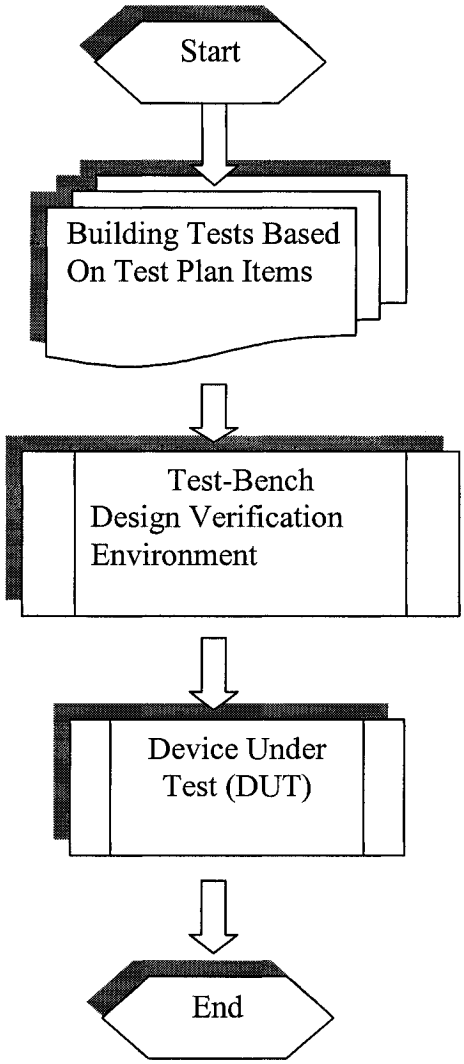

Figure 4: Deterministic Testing Flow Chart

3.2.2 Coverage-Driven Functional Verification

Coverage-Driven Functional Verification is the solution to today verification challenges. The DVE (Design Verification Environment) is built on Designs' Functional Specifications by implementing the Functional Test Plan and pointing out all the functional coverage items that need to be covered to indicate the effectiveness and the completeness of the verification task. Usually the DVE is designed on the concept of random generation with qualified and permitted set of constrains. Bus functional models and drivers in the DVE instantiate the simulation by issuing valid transactions against the Device Under Test. Functional coverage is collected and based on the completeness ratio, new deterministic tests are built to fill the coverage holes. Once all the coverage holes are covered and a 100% coverage is accomplished, only then the verification effort can be considered as complete. Building the Design Verification Environment on the concepts of Functional Coverage will:

- Minimize the number of deterministic tests required to verify the DUT.
- Reduce the simulation time.
- Determine when the verification task is complete.
- Reduce the hassle of modifying the DVE as a result of design changes.
- Enhance overall the verification process.

The following DVE diagram [48] and flow chart shows the Coverage-Driven Functional Verification Approach.

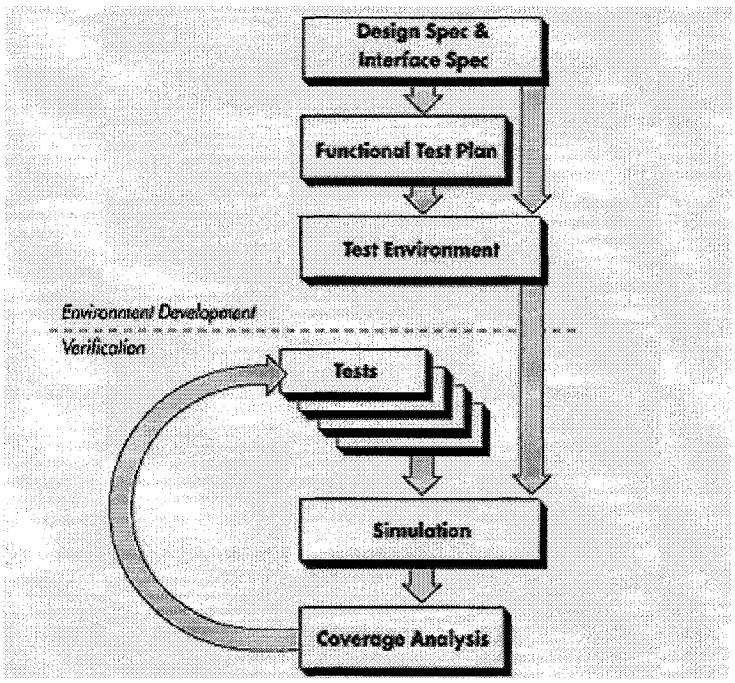

Figure 5: Coverage-Driven Functional Verification Approach

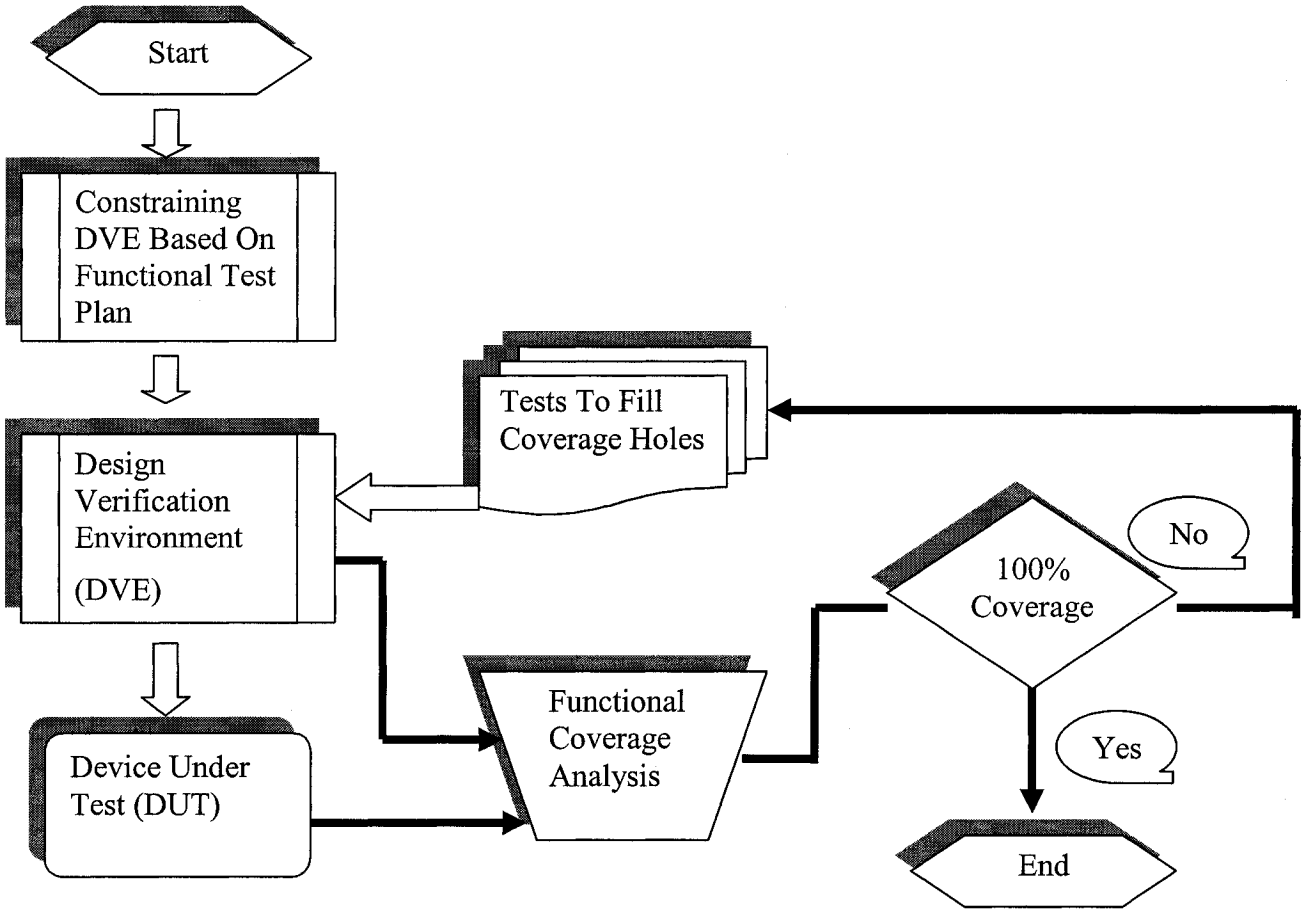

Figure 6: Coverage-Driven Functional Verification Flow Chart

In order to exercise different aspects of the design, different inputs or tests need to be fed into the system. For most systems, creating these tests has become very challenging; it involves creating extremely sophisticated data, with parallel stimulus from various input streams, etc. To simplify test creation, automatic test generation was developed to substantially reduce the time it takes to create an effective test. This automated approach allows users to express high-level intentions, which allows the test-bench automation tool to create various stimuli that manifest those intentions. By reviewing the input that was created, the user can then direct the generation towards areas that are yet to be exercised to ensure completeness.

Historically, a few coverage metrics have been used to evaluate the verification process, but all fail to address the aforementioned needs. A more suitable solution for these hard problems is functional coverage measurement [48].

3.2.3 Functional Coverage Approach

Functional coverage perceives the design from a user's or a system point of view. It covers all the typical DUT scenarios, error cases, corner cases, protocols, and relationships between scenarios: for example having interrupts while visiting all the states of a state machine or having all lengths of packets while injecting errors randomly among them and etc.

Using Specman Elite Coverage Reports, the hardware details are hidden from the verification engineers. It basically builds a higher level of abstraction where signals names and bits vectors in the design are abstracted and instead the verification engineer deals only with high-level terms (collection Functional items) that correspond to Test Plan Items.

3.2.3.1.1 Functional Coverage Tool Requirements

Coverage-Driven Functional Verification is the right solution to today Verification complexities and problems, however as it is mentioned in Verisity white paper "Coverage-Driven functiona Verifcation"[48] ,certain requirements are imposed on the Functional Coverage Methodology (Tool):

- ***Expressiveness of Queries:***

A query should be easy to specify.

- ***Informative Coverage Reports:***

Coverage results should be readable and intuitive.

- ***Coverage Feedback:***

A coverage feedback should be provided to indicate which adjustments are needed to improve the coverage results.

- ***Test-base Ranking:***

There should be a mechanism to accumulate and analyze coverage reports from multiple simulation runs.

- ***Timing of Analysis:***

The coverage tool should allow the engineer to analyze the coverage information both between simulations and during a test run.

- ***Grading:***

There should be a mechanism to assign weights to different functional coverage items. Different coverage scenarios can be distinguished by setting specific weights.

- ***Optimizing the test-suite:***

There should be a ranking mechanism to allow us to create a subset of tests that verify the DUT to a significant degree with a minimum amount of resources. Running this subset of tests instead of your entire test suite drastically reduces the total number of cycles needed for verification.

- ***Open Environment:***

It is critical to have a flexibility to use the same methodology on all types of designs.

3.3 Performance Measure and Statistical Analysis

It is shown from ASIC Verification experiences that Coverage-Driven Functional Verification and Reuse Methodology has a much better performance than Deterministic Testing. In our thesis, applying the coverage driven functional verification methodology, we discovered bugs that they were not discovered by the deterministic testing approach. The issues found could be minimized and avoided with Coverage-Driven Functional Verification. Basically early in the design stages the coverage items are outlined based on the ASIC Design Specification. With Random constraints in the test bench (i.e. very few test cases) the simulation is run and coverage reports are collected. If there are illegal scenarios found, then directed test cases can be written and the regression can be run again to see if the new fix done by the RTL designer is correct and nothing else is affected. Regressions are done very quickly. It takes few hours, less than a full working day.

Comparing both verification methodologies we can see that with Coverage-Driven Functional Verification, there is a simulation performance enhancement of 4 times than the deterministic testing one. The two diagrams below show the number bugs found in the design, the number of tests required using both the deterministic approach and the Coverage-Driven Functional approach.

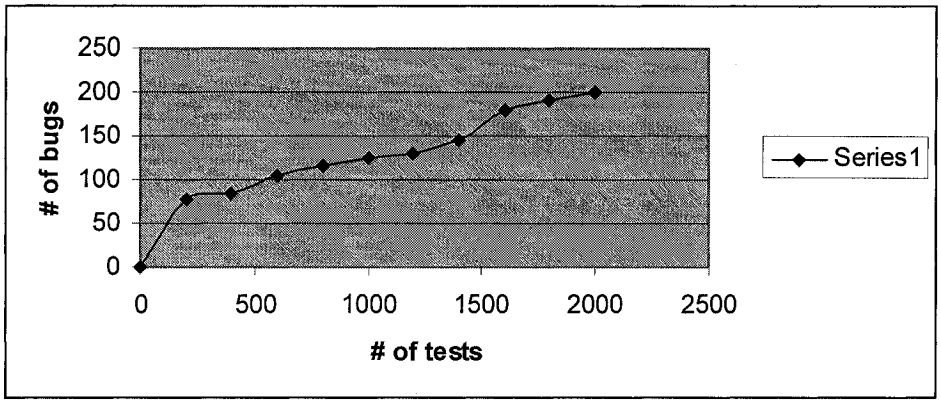

Figure 7: Deterministic Testing Approach: # of tests vs. # of bugs

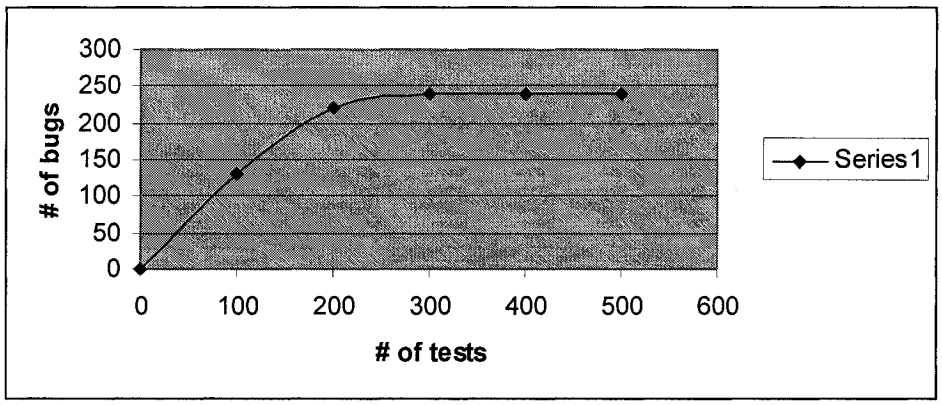

Figure 8: Coverage-Driven Functional Verification Approach: # of tests vs. # of bugs

In figure 8, it is noticeable that many tests were written to find 200 bugs and a decision was made to stop writing any more tests because of time to market and the stability of the already written tests over number or regressions ran without errors. However if more functionalities and random stimulus were applied, more bugs might appear. So there is no 100% assurance that the design is free of errors.

In figure 9, Coverage driven was applied to another ASIC of the Linecard with similar number of gates and complexity. As it shown from the start of the simulations, bugs were found quickly because the test bench was built on randomness and constrains to cover every item of the ASIC's Functionalities. Few hundred directed test cases were written to cover an almost impossible scenarios / corner cases. After writing 250 test cases all the errors were found. Additional tests were written to cover certain scenarios which were impossible to cover using only constraints such as injecting errors in registers and etc. With many regressions, the errors count was still flat around 240 bugs. In addition to that more errors and stability were found using Coverage-Driven Functional Verification Approach.

So using Coverage Driven-Functional Verification (CDV) Approach, we assumed to get the following performance measurement.

Simulation Time:

CDV = 4 times faster than the DT (Deterministic Testing Approach).

DT = ¼ CDV.

Bugs Found:

CDV, # of bugs found = 240 bugs in 1.2 year.

DT, # of bugs found = 200 bugs in 2.5 years.

Schedule Date

CDV, time to market = did not slip any schedule.

DT, time to market = slipped schedule and caused 1 re-spin.

Mathematical Formulas:

CDV Approach:

- Let us assume that "T" represents a Test Case.
- Let us assume that "n" represents the number of Test Cases.
- Let us assume that "C" represents the whole coverage items.
- Let us assume that "F" represents the functional coverage item.
- Let us assume that "S" represents the whole simulation for the functional coverage.

$$C = \sum_{i=1}^{n} Fn$$

$$S = Time\,(C) + Time\,(\sum_{i=1}^{n} Tn)$$

DT Approach:

- Let us assume that "T" represents a Test Case.
- Let us assume that "n" represents the number of Test Cases.

- Let us assume that "D" represents the whole set of Test Cases.
- Let us assume that "S" represents the whole simulation Time.

$$D = \sum_{i=1}^{n} Tn$$

S = Time (D), where D could be in 1000s.

3.4 Design Example and Verification using CDV and RM (Reuse Methodology)

3.4.1 Design Example

Let us assume a small design (DUT) which has 5 buffers. Each buffer size is 256 bytes. The DUT receives packets of maximum size 256 bytes. So the DUT can buffer up to 5 maximum size packets. This DUT has a memory controller interface. It issues requests to write packets to memory then upon acknowledgement from the memory controller, it writes packets to memory. The memory bus width is 64 bytes. So a whole packet (max. size) can be written by sending 4 bursts. There is a command bus which decides the type of transmission with the memory controller. There is even parities for both data / address bus and command bus.

3.4.1.1 Verification Task

- Developing a design verification environment (DVE) on the concepts of using Deterministic Testing as the verification process. A test plan will be written and implemented by the deterministic testing approach. The test plan is derived from the DUT functionalities.
- Developing a design verification environment (DVE) on the concepts of using Coverage-Driven Functional Verification and Reuse Methodology. A test plan will be developed on the concept of coverage items collection based on the DUT specifications and functionalities.

3.4.1.1.1 DVE for Deterministic Testing Approach

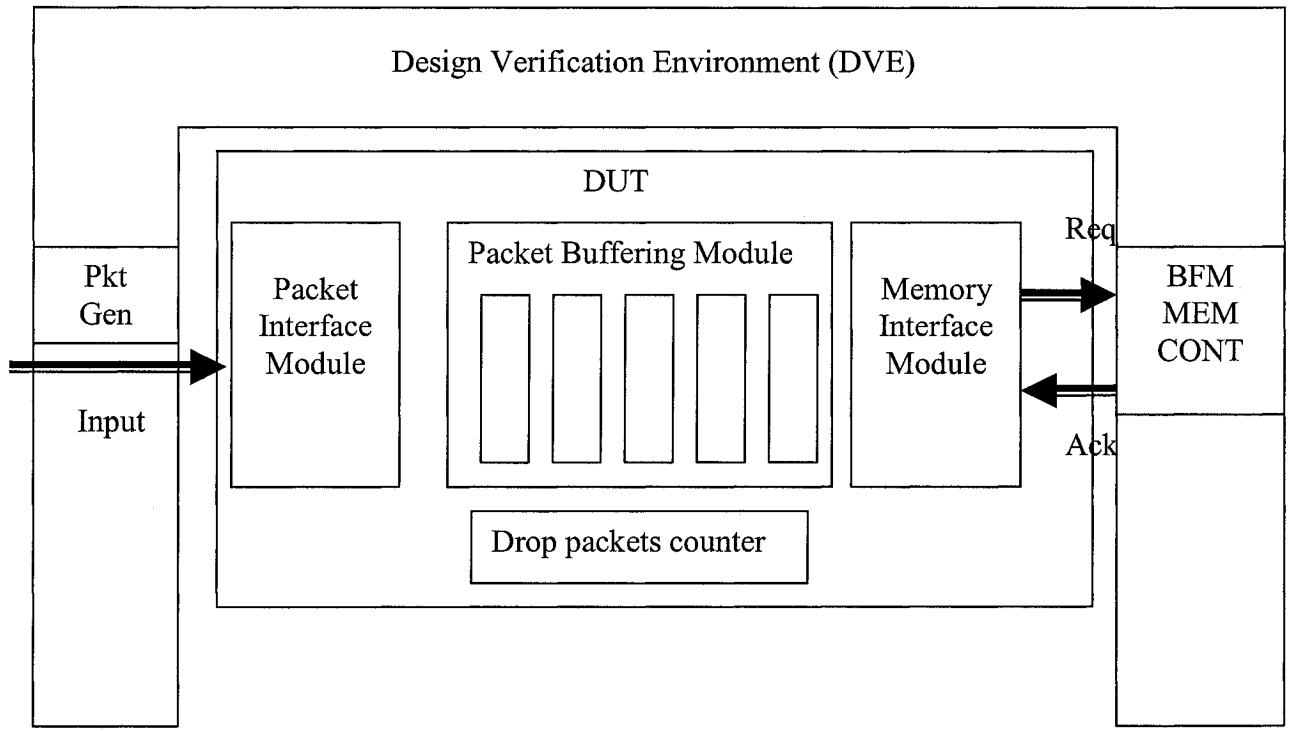

Figure 9: Example Design Verification Environment

3.4.1.1.1.1 Test Plan

Below are examples of few test cases for the design.

- Generate large packets from 150 bytes to 250 bytes incrementing by 10 bytes at the time.
- Generate medium packets from 80 bytes to 150 bytes incrementing by 10 bytes at the time
- Generate small packets from 1 byte to 80 bytes incrementing by 10 bytes at the time.
- Repeat the steps above while changing the incremental bytes by 5 bytes.
- Transmit a mix of small, medium and large packets.
- Send 5 packets, each 256 bytes.
- Send 6 packets, 4 packets 256 bytes, 1packet 255 bytes and the last packet 1 byte. Repeat the same test with the last packet 2 bytes or more.
- Send requests to memory and ack all of them and check if the packets are written to memory.
- Nack all the DUT memory write requests and check if the drop counter is incremented correctly.
- Check the maximum drop counter size and apply reset and verify that the counter is set to zero.
- Etc.

3.4.1.1.1.2 Deterministic Testing Approach: Test Cases

Each point of the test plan is implemented in a test case. Sometimes there is a need for more test cases than test plan points (i.e. sometimes a test plan point is implemented with 2 or 3 test cases to cover it). So if we assume there is 20 test plan points for this small design, it might require 30 test cases. Notice that only the test plan is implemented. No randomness is applied to catch corner cases or hidden bugs.

3.4.1.1.2 DVE for Coverage-Driven Functional Verification and Reuse Methodology

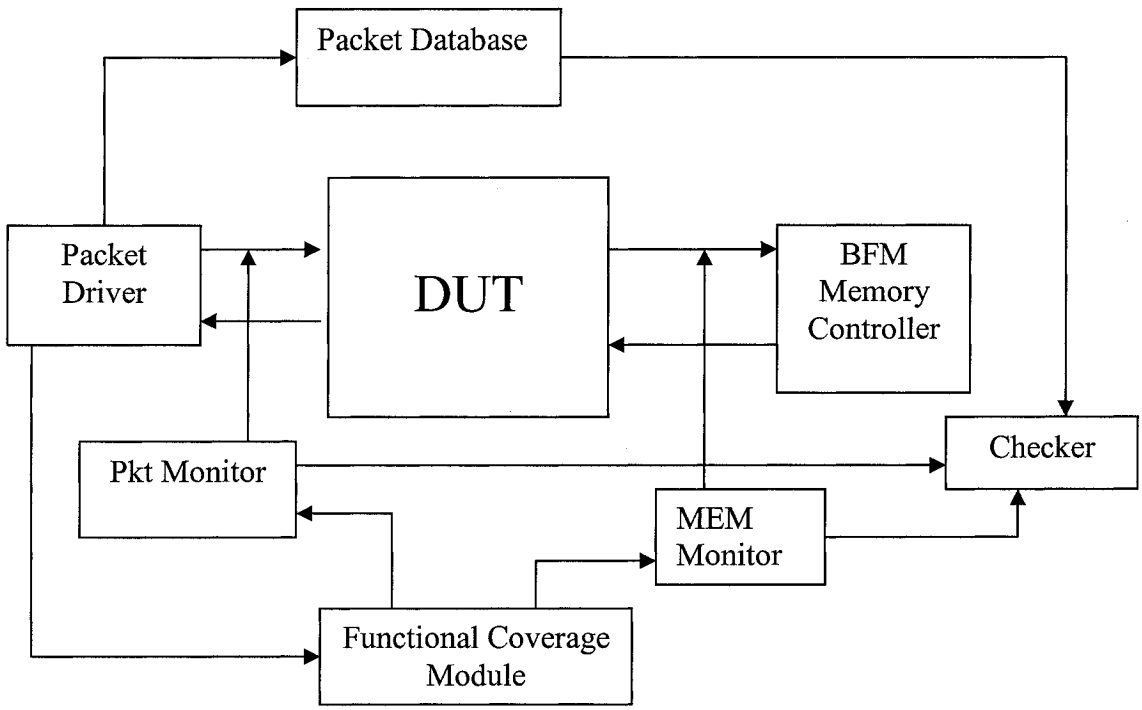

Figure 10: Design Verification Environment

3.4.1.1.2.1 Functional Coverage Test Plan

- Define buckets to collect allowed packets sizes: bucket #1 : 1 to 80 bytes, bucket #2: 80 to 150 bytes, bucket #3: 150 to 256 bytes.
- Define an item when there is 5 packets of size 256 bytes are sent.
- Define items (Coverage items) to collect information on all Reqs, Acks, Nacks scenarios.
- Define a coverage item for the drop counter = 0, 256, 0 after reset.
- Etc.

3.4.1.1.2.1 Coverage-Driven Functional Approach and Reuse Methodology

- The DVE is built on the concepts of E verification components and encapsulated in a package which can be used, integrated (ported) into other ASICs or system verification environments.
- The Functional Coverage module is based on certain events derived from the monitors. The coverage items are updated and once there is an illegal action, an error is reported.
- Coverage reports are generated and based on the results, directed tests are written to cover the coverage holes.
- The Design Verification Environment is built on the concepts of random generation. So running the verification environment alone for few times with different seeds would collect coverage items and eventually bugs in the DUT are discovered.
- All the DUT errors are discovered with fewer test cases and less simulation time.
- The concept of sequences in Specman eRM (e Reuse methodology) is used to force object-oriented approach, reusability and enhance the verification process. Reusability is crucial in designs development cycle. Since most of the designs are based on previous ones except with additional features and functionalities, design and verification environment reusability are vital to the success of any project since winning competition are time to market are met by taping out the design with less time. With reusability we only need to add or modify existing modules of the verification environment. So within few weeks the verification environment is ready to debug RTL code and design.

Chapter 4

4 Ethernet IP Core Verification

In this chapter both deterministic and coverage-driven functional verification and are applied. The deterministic testbench approach was applied by the OpenCores and it will be shown in this section. Both the Ethernet IP Core Specification and deterministic testbench approach will be taken from OpenCores documents and explained in this section [34]. Also in this section, I will be going in details over my coverage-driven functional verification.

4.1 Architecture

The Ethernet IP Core consists of 5 modules:

- Host Interface and the BD structure
- TX Ethernet MAC (transmit function)
- RX Ethernet MAC (receive function)
- MAC Control Module
- MII Management Module

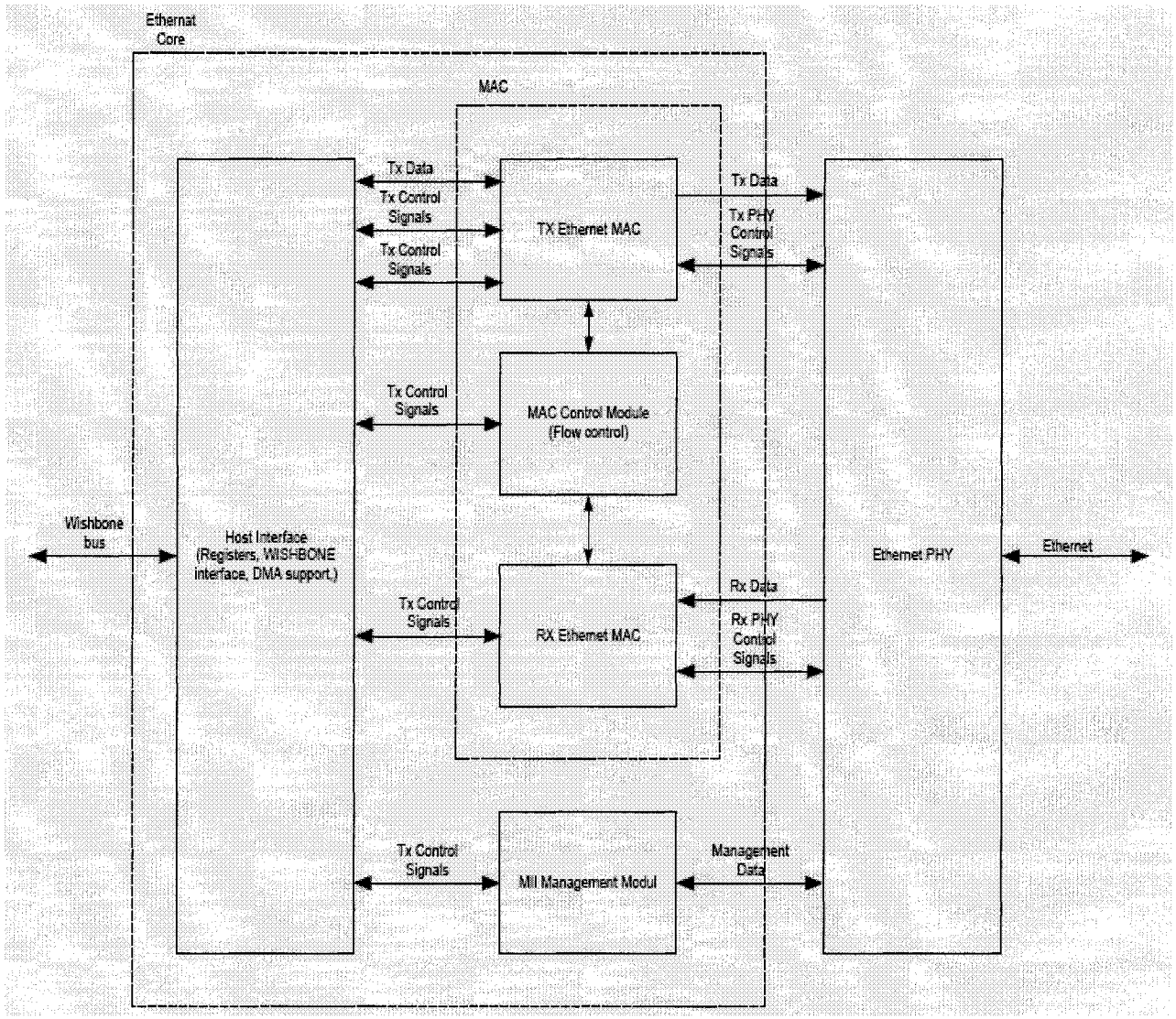

Figure 11: Architecture Overview

4.2 Ethernet MAC IP Core OpenCores Deterministic Testbench

4.2.1 Overview

Ethernet MAC IP Core testbench consists of a whole environment for testing Ethernet MAC IP Core, including Ethernet PHY model, WISHBONE bus models with bus monitors and test cases, which use those models to stimulate transactions through the Ethernet. Those transactions are checked in many different modes.

4.2.2 Testbench File Hierarchy

The hierarchy of modules in the Testbench of the Ethernet MAC IP Core is shown here with file tree. Each file here implements one module in a hierarchy. Source files of the Testbench are in the ethernet\bench\verilog subdirectory.

. tb_ethernet.v

. eth_phy.v

. wb_bus_mon.v

. wb_master_behavioral.v

. wb_master32.v

. wb_slave_behavioral.v

. wb_model_defines.v

. tb_eth_defines.v

. eth_phy_defines.v

4.2.2.1 Testbench Module Hierarchy

Module hierarchy is shown in detail in the following picture. Description of modules and their connections is described below.

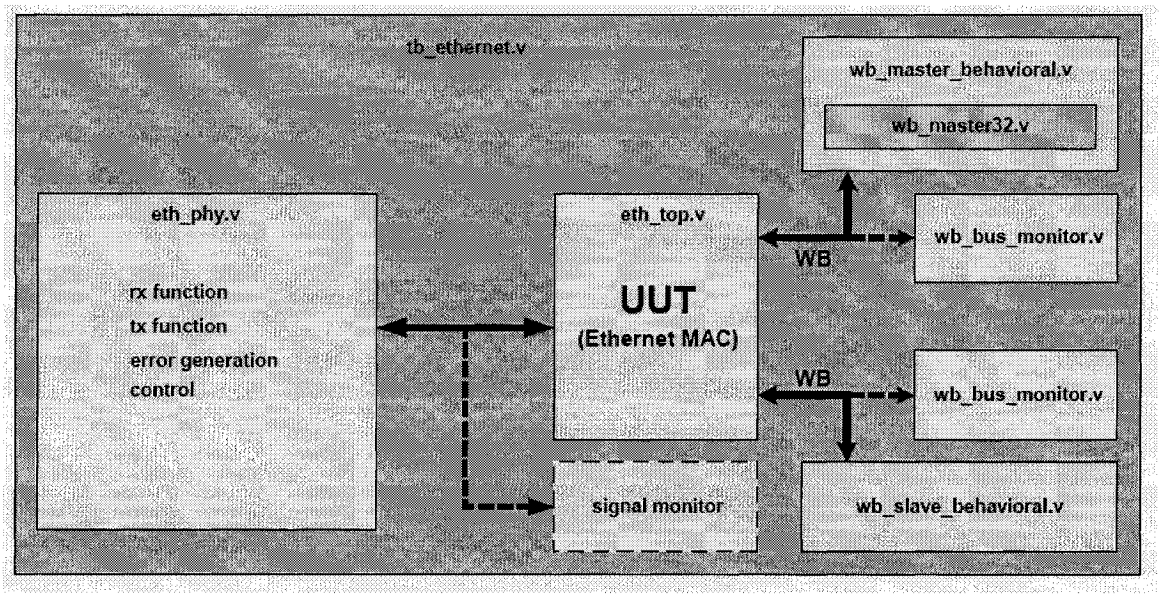

Figure 12: Test Bench Module Hierarchy

4.2.3 Description of Testbench Modules

The module tb_ethernet.v is used as testing environment and it incorporates beside all test submodules, functions and tasks also Unit Under Test (Ethernet MAC IP Core). Description of tasks is covered in chapter Description of Testcases, while all test submodules are described in the following chapters.

4.2.3.1 Description of Ethernet PHY module

Ethernet PHY module simulates simplified Intel LXT971A PHY chip. Ethernet PHY provides two clock signals to the Ethernet MAC Core: transmit clock (mtx_clk_o) and receive clock (mrx_clk_o). Depending on the control bits, TX and RX clock operate at 2.5 MHz for 10 Mbps operation or 25 MHz for 100 Mbps operation (only bit [13] is used for clock frequency setting). TX and RX clock signals are not synchronous. When Ethernet link is not up, RX clock has a random frequency between 2 MHz and 40 MHz. PHY has an MIIM interface, which is connected to the Ethernet core. All transactions are monitored and every error/warning reported. Besides that PHY

has several registers implemented in it (Control, Status and two Identification registers). PHY provides carrier sense and collision signals. Both signals can be set through several tasks. When transmitting data (PHY is receiving data), PHY controls the protocol (preamble, sfd, writes length and data to its memory). When PHY sends data to the Ethernet MAC, it can generate various preambles (different length, wrong preamble). It takes data from its memory. Testbench needs to write data to PHY's memory before PHY can start with transmission.

4.2.3.2 Description of WB submodules

4.2.3.2.1 wb_bus_monitor submodule

The module **wb_bus_monitor.v** monitors the WB Bus and tries to see WB Protocol Errors. There are two point-to-point WB buses:

- WB master from Ethernet MAC IP Core that goes to the WB Slave Behavioral unit (for writing and reading data)
- WB slave from WB Master Behavioral unit to the Ethernet MAC IP Core (used for accessing registers and buffer descriptors).

There are also two WB bus monitors, one for each WB bus.

4.2.3.2.2 wb_master_behavioral submodule

The module **wb_master_behavioral.v** is used to initiates WB cycles to WB Slave in the Ethernet MAC IP Core. That is controlled by top-level. This module also includes a submodule **wb_master32.**v, which is used to generate proper WB cycles. The length and type of each cycle is controlled by **wb_master_behavioral.v** module. This module also incorporates a block of SRAM.

4.2.3.2.3 wb_slave_behavioral submodule

The module **wb_slave_behavioral.v** responds to cycles initiated by WB Master in the Ethernet MAC IP Core. When to respond and a type of cycle termination is controlled by top-level. This module also incorporates a block of SRAM.

4.3 Coverage-Driven Functional Verification and Reuse Methodology Environment

4.3.1 Overview of Specman (e) Language

Specman e language is a powerful object oriented language created for verifying complex ASICs designs. It is very popular and used widely by most of the Hi-tech companies. The power of the Specman language is its random generation engine, functional coverage collection / reports, eRM (e Reuse Methodology), eVC (e Verification Components) and a well-defined verification methodology. I chose the Specman language because it is so powerful, used widely across my company and I have an easy access to its tools. Most importantly using Specman Language, I can prove my point that Coverage-Driven Functional Verification and Reuse Methodology are the way to verify complex ASIC designs, reduce time to market and produce bugs free designs. Using its random generation engine, functional coverage and eRMs, I will be able to verify the Ethernet Mac IP Core and show its advantages over the OpenCores Verilog Deterministic (Directed) Testbench.

Below is a brief description of Specman E language from Verisity Inc.[48]

Functional verification is the bottleneck in delivering today's highly integrated electronic systems and chips. Verisity's Specman Elite® gives you the industry's most powerful capability for automating the process of verification. Yesterday's methods of verification just can't keep pace with today's design verification realities. Only a few iterations for debugging are acceptable as time-to-market windows shrink. To achieve first time success you can no longer hope that mass quantities of simulation or emulation uncover all underlying problems. Verisity's Specman Elite offers you a comprehensive environment for all aspects of verification: automatic generation of functional tests, data and temporal checking, functional coverage analysis, and HDL simulation control. You may already have invested time and effort on internal solutions, but realistically these ``tools" are rewritten project to project without allowing for significant reuse. They also don't contain the engines already built into Specman Elite, such as Verisity's patented Constraint Solver, which allows fast and easy test generation-built in! With Verisity's Specman Elite, you capture the rules from the specifications and use this info to automate the functional verification process. Specman Elite's methodology finds the "bugs you haven't thought of" in your Verilog or VHDL design -- caused primarily by ambiguities in the spec, or unanticipated usage by the target system. The result is faster verification and higher quality products. Get working silicon to market on time!

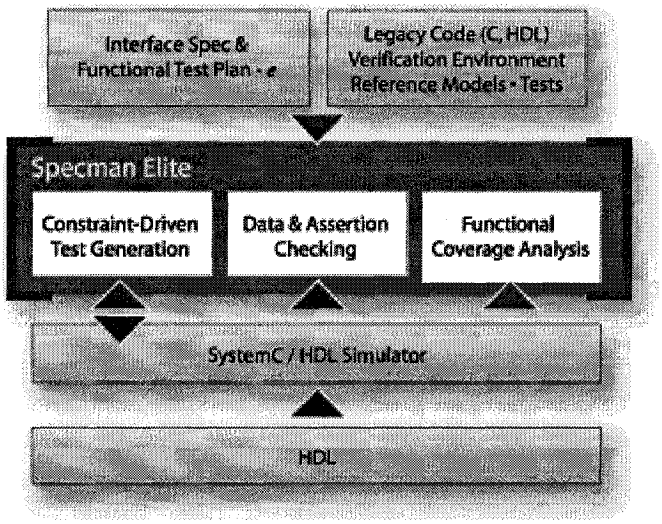

Figure 13: Specman Elite entire process of verification from block, chip and system level, and project levels

Our powerful verification language e allows you to capture the rules from specifications as well as generate tests automatically. Specman Elite eliminates misinterpretation of specifications. With Specman Elite's constraint-driven test generation, you can now automatically generate tests for functional verification. By specifying constraints, you can quickly and easily target the generator to create any test in your functional test plan. These tests can even be generated on-the-fly based on the current design state, making it possible to generate even hard-to-reach corner cases. Powerful temporal constructs enable you to capture complex protocols for checking. On-the-fly data checking and generation allows context-specific expected values. You can use any combination of gray/black/white box checking to speed debugging.

An executable functional test plan measures the progress of verification. Functional analysis automatically identifies holes in the test coverage. Verification schedules become more predictable because functional coverage is a meaningful and direct measure of the completeness of your verification. All leading HW/SW co-verification tools are supported. For example, Mentor Graphics' SeamlessT co-verification environment is deeply integrated to enable functional testing of both hardware and software. Early integration and debugging of hardware /software systems will eliminate errors and shorten time to market for the combined system. [48]

4.3.2 Benefits of Using Coverage-Driven Functional Verification

4.3.2.1 Benefits of Coverage-Driven Functional Verification

Using Coverage-Driven Functional Verification and Reuse Methodology is a must have verification environment to verify multi-million gate ASIC designs. It is impossible to completely verify complex ASIC designs with directed tests and finite number of tests. Writing thousands of directed-tests require many man

powers and time. Projects hardly can be successful with this verification approach. With Coverage-Driven Functional Verification based on the power of Specman Random generation engine, we could develop a complete verification environment which can discover every functionality of ASIC designs. Adding to that with the Reuse Methodology, we can reduce the time of development by reusing existing verification components in Chip and System Level Verification which enhance the verification process and reduce crucial verification time. Knowing that the design will be fabricated and ready for software drivers' development without any bug is the milestone of any company and design group. Having bugs after the chip comes back from the fab is a huge disappointment, major failure and loss of business. That is why verification is the most important and time consuming part of the design development cycle.

Now emphasis are always put on verification and mainly using the right and most efficient verification methodology. Building Design Verification Environments (DVE) using Coverage-Driven Functional Verification and Reuse Methodology is the right approach, however the time for developing the DVE is the most consuming part in the verification cycle and once the environment ready, the amount of bugs found in the design is very large and it is done in a limited time. The text and diagram below written by YOGITECH SPA shows this fact:

"In the system-like verification, coverage from each eVC will give a complete evaluation for all the interfaces and then the user will be able to extend his test programs to cover what was not tested.

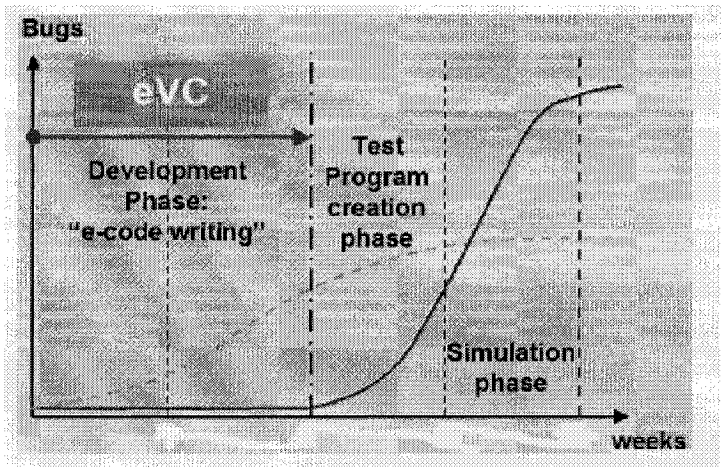

Figure 14: eVC Verification timeline

Past experiences show that time requirements for medium complexity module verification are about 2 man weeks when a reusable environment is available (eVC, test sequences and documentation) against the 5-6 man weeks starting from scratch and lower functional coverage." [26]

The following text below written by "**The *e* Language: A Fresh Separation of Concerns**" paper shows the importance of functional driven verification:

"A few words are needed here to explain the term *functional verification* as used by the design automation community. The target of this verification activity are hardware devices including CPUs and other microchips, network equipment such as routers, and systems combining hardware with embedded software. Functional verification is the process by which the correctness of the hardware design is established, the term *testing* having long been associated in this community with the process by which *manufacturing defects* are detected. In functional verification, the hardware is not tested directly. Instead a description of the hardware written in a hardware description language (HDL) such as VHDL or Verilog, is compared with a different, high level description of the hardware. Software written in ***e*** may therefore play a double role. On the one hand it is used for expressing the expected behavior of the *device under test* (DUT). On the other hand, it is used for exercising a certain functionality of the DUT". [48]

Another paper "**Object Oriented Hardware Synthesis and Verification"**

"The generation process ensures non-zero probability for any legal input sequence. However, given the huge input space and state space of modern devices, a practical approach would control the distribution of input sequences in view of the accumulated coverage. Functional coverage points are user defined combinations of states, or sequences of states that have some architectural or micro-architectural significance. Because of its sequential nature, functional coverage is a more rigorous metric than code coverage. The accumulated functional coverage and its breakdown to architectural and micro-architectural features provide status information about the verification effort. This information is used for steering the process and to eventually certify that the DUT has a high probability of being functional." [26]

A text from Verisity Functional Coverage Analysis

"An executable functional test plan measures the progress of verification. Functional analysis automatically identifies holes in the test coverage. Verification schedules become more predictable because functional coverage is a meaningful and direct measure of the completeness of your verification."[48]

"An executable functional test plan measures the progress of verification, and functional analysis automatically identifies holes in the test coverage. Since functional coverage is a meaningful and direct measure of the completeness of your verification, Specman Elite's functional coverage analysis increases predictability in your verification schedules." [48]

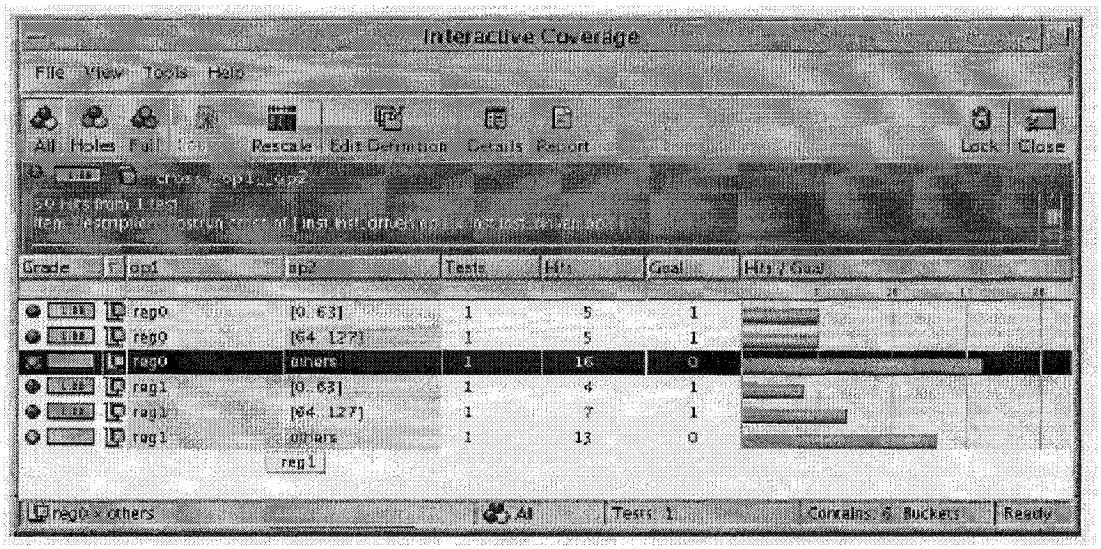

Figure 15: Functional coverage analysis allows you to identify holes and direct the entire process

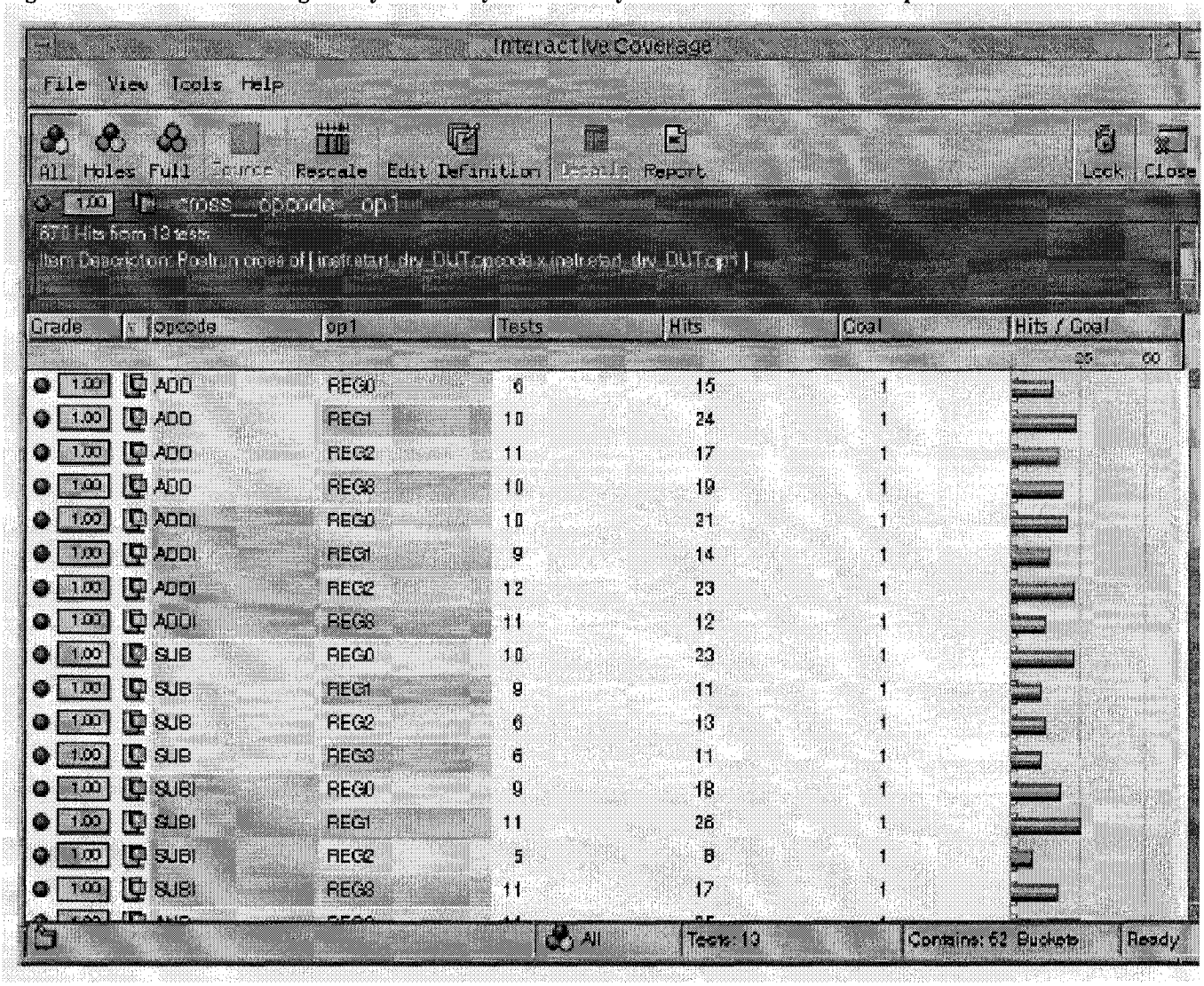

Figure 16: Specman Functional Coverage Report

4.3.3 Ethernet Mac IP Core Coverage-Driven Functional Verification and Reuse Methodology Design Verification Environment

4.3.3.1 Introduction

The Coverage-Driven Functional Verificaiton and Reuse Methodology verification environment is developed to verify the Ethernet MAC IP Core from OpenCores. The Ethernet IP Core is a MAC (Media Access Controller). It connects to the Ethernet PHY chip on one side and to the WISHBONE SoC bus on the other.

As it is mentioned in the Ethernet IP Core document [34], the following lists describe the main features of the Ethernet IP Core:

- Performing MAC layer functions of IEEE 802.3 and Ethernet.
- Automatic 32-bit CRC generation and checking
- Delayed CRC generation
- Preamble generation and removal
- Automatically pad short frames on transmit
- Detection of too long or too short packets (length limits)
- Possible transmission of packets that are bigger than standard packets
- Full duplex support
- 10 and 100 Mbps bit rates supported
- Automatic packet abortion on Excessive deferral, too small inter packet gap, when enabled
- Flow control and automatic generation of control frames in full duplex mode (IEEE 802.3x)
- Collision detection and auto retransmission on collisions in half duplex mode (CSMA/CD protocol)
- Complete status for TX/RX packets
- IEEE 802.3 Media Independent Interface (MII)
- WISHBONE SoC Interconnection Rev. B2 and B3 compliant interface
- Internal RAM for holding 128 TX/RX buffer descriptors
- Interrupt generation on all events

The Coverage-Driven Functional Verification and Reuse Methodology Verification environment consists of three eVCs : PHY eVC, Wishbone eVC, and Ethernet eVC. Each eVC is developed independently so it can be reused later in any other project. Then the three eVCs are merged under one environment to verify the Ethernet MAC IP Core design (DUT). The e Verification Component (eVC) is a ready-made, highly configurable e verification environment suitable for verifying thoroughly the designated design under test. Each eVC of the Ethernet MAC IP Core Verification environment consists of the following features:

- Predefined generation, checking, and coverage capabilities.
- Custom generation, checking and coverage capabilities for special DUT needs.

- Full compliance with Verisity's e Reuse Methodology (eRM), in particular the eRM architecture.

An *e*VC. is an *e* Verification Component. It is a ready-to-use, out-of-the-box verification environment, typically focusing on a specific protocol or architecture, such as Ethernet, PCI Express, AHB, PCI, or USB.

Each *e*VC consists of a complete set of elements for stimulating, checking, and collecting coverage information on your device under test (DUT). The *e*VC expedites creation of a more efficient test bench for your DUT. The *e*VC can work with both Verilog and VHDL devices and with all HDL simulators that are supported by Specman Elite. You can use an *e*VC as a full verification environment or add it to an existing environment. [49]

There are two types of sequence's transactions in any eVC which is eRM compliant: Pull Mode and Push Mode. The push mode is when the sequence driver keeps pushing sequence's items to the lower layer "BFM". Once all the iterms are transmitted, the simulation stops (i.e. there is no more transactions for the BFM to send to the DUT). In pull mode, all the control is in the lower layer "BFM". The BFM keeps asking for more items form the sequence driver "Upper Layer" and once the BFM is finished sending the transaction to the DUT, he issues another command "item done" to the sequence driver to inform him that he is done with the current transaction and he is requesting next item. The sequence driver will not send another transaction until he sees the item done command. Once the BFM is finished sending the transactions to the DUT, it stops the simulation.

In the Ethernet Verification Environment, the coverage-driven functional verification is used as a metric-driven verification process that includes total coverage measurement and analysis, and a means of assessing functional verification progress. The Ethernet eVCs are optimized for use in a coverage-driven verification process by providing:

- Random generation to cover all the possible behaviors of the design
- Constraints user interface to narrow the generation in a way to create directed test cases.
- Self checking
- Coverage measurement

In all the Ethernet eVCs, e reuse methodology is used to enable major productivity gains in functional verification. eRM provides best practices to enable reusable, consistent, extensible, and plug and play verification environments.

The Ethernet eVCs have been created in compliance with eRM guidelines, providing:

- Predefined agents, monitors, and BFMs built to work with the Ethernet protocol
- Predefined sequences, allowing to build easily test scenarios for DUT
- Predefined checks and coverage groups

- Log Messaging

All the Ethernet IP Core eVCs have common terminology. These terminologies are:

- Design Verification Environment: a collection of eVCs accompanied with their verification components.
- Monitor: a unit instantiated in the verification environment to monitor passively the DUT signals, trigger events and collect data for integrity checking.
- Agent: a unit instantiated in the verification environment that models a device as an active agent or monitors a DUT as a passive agent. When it is an active agent, it drives DUT signals. When it is a passive agent, it monitors the activity of DUT agent. Monitors, checkers, scoreboards, and coverage are part of the agent.
- Bus Functional Models (BFM): a unit instantiated in the verification environment that interact directly with the design (DUT) and drives or samples the DUT signals.
- Data Item: a struct instantiated in the verification environment to represent DUT's input such as data transaction, a packet and etc.
- Sequence: a struct instantiated in the verification environment to represent a stream of items signifying a high-level scenario of stimuli.
- Sequence Driver: a unit instantiated in the verification environment to drive the sequences into next layer, BFM, to be driven to the DUT.

Each of the Ethernet IP Core eVCs can generate traffic stimuli to the DUT, check that the DUT adheres to the protocol and lastly but not least collect functional coverage of the DUT.

Each eVC will have a library of sequences to generate many interesting scenarios to test thoroughly the Ethernet MAC IP Core design. The eRM and sequences define a Test Writing Interface where RTL designer can use to define their user-defines sequences to verify the DUT. Basically test writers can use the eVC to write efficient tests while treating the eVC as a block box and only a well defined test writing interface is outlined clearly for them. They can build sequences by constraining the driven sequence's items and add these sequences to the sequence library. Having this well defined test writer interface will help in having the RTL designers writing test cases and help in the verification efforts and to meet time to market. Writing tests (sequences) is so easy that the RTL designers don't need to know the Specman e language. A certain test sequence template will be generated where they have only to change / constrain differently the values of the sequences.

A common verification environment (VE) directory structure will be adapted in all the eVCs. Figure 21 shows an eVC directory structure. Below is a description of each directory:

- e : contains the e sources that make up the verification environment.

- seq_lib: contains the e sources for user-defined sequences.
- sim: contains the simulation results.
- scripts: contains the scripts files to run simulation.
- tb_hdl: contains the HDL testbench top files.
- tests: contains DUT simulation testcases.
- doc: contains design verification documents such as Test Plan and DVE.

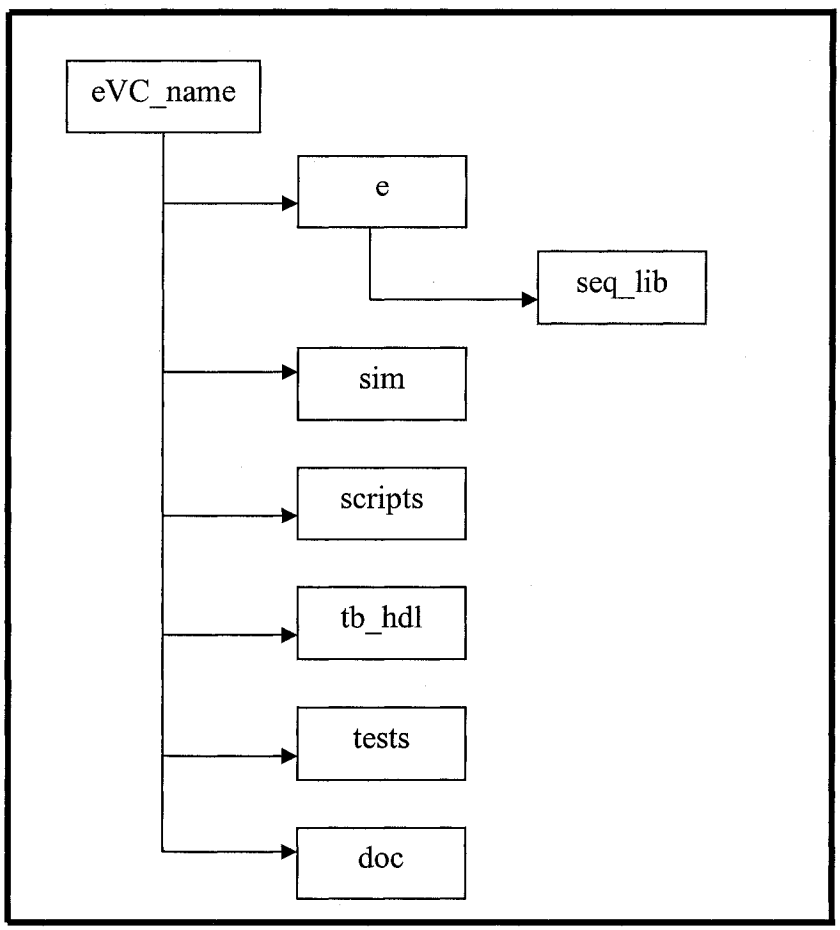

Figure 17: Verification Environment (VE) Directory Structure

4.3.3.2 Ethernet PHY eVC

The Ethernet PHY eVC is a complete e verification component and it is an eRM compliant. It consists of all the modules required to be integrated into a system level verification environment or used as a standalone environment. It is implemented to a model a simplified version of the Intel PHY LXT971A chip.

The PHY eVC models a single-port Fast Ethernet 10/100 transceiver that supports 10Mbps and 100Mbps networks and complies with all applicable requirements of IEEE 802.3. It will implement a Media Independent Interface (MII), clock generation logic, PHY registers, wishbone (CPU) interface, Cyclic Redundancy Check (CRC) logic, Rx/TX state machines logic, memory, drivers, counters, data generation / manipulation.

4.3.3.2.1 Ethernet PHY eVC Architecture

As it is shown in Figure 18: Ethernet PHY eVC architecture, the Ethernet PHY eVC consists of the following modules:

- PHY environment unit
- PHY active agent unit
- PHY passive agent unit
- PHY fields configuration unit
- PHY signals mapping unit
- PHY MII sequence driver and sequences
- PHY data sequence driver and sequences
- PHY Ethernet BFM
- PHY monitor / checker
- PHY functional coverage

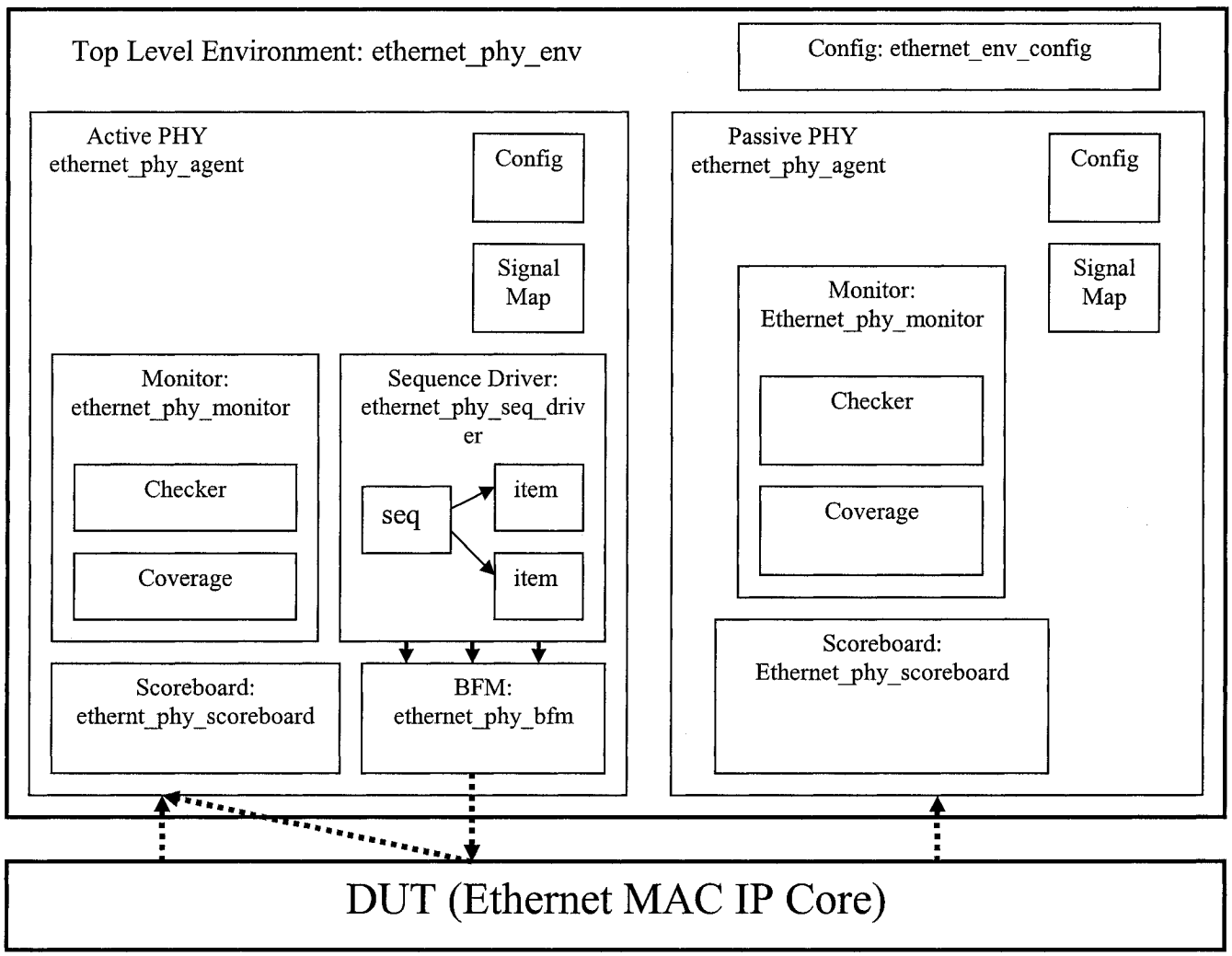

Figure 18: Ethernet PHY eVC Architecture

The Ethernet PHY eVC is represented by ethernet_phy_env.

The eVC simulates the PHY behavior of the Media Independent Interface (MII).
The active PHY ethernet_phy_agent unit simulates the PHY behavior. The eVC verifies the DUT with the active PHY agent generating traffic to the DUT and responding to its traffic. The passive agent does not drive anything to DUT, however it passively monitors the bus signal interface between the DUT and PHY.

The Ethernet PHY active agent consists of:

- A sequence driver "ethernet_phy_seq_driver", and a BFM "ethernet_phy_bfm".
- A monitor "ethernet_phy_monitor" to do DUT's checking and collects functional coverage.
- A scoreboard "ethernet_phy_scoreboard" to compare the collected DUT's and expected data.

All the active transactions with the DUT are done by the BFM. The BFM is a driver which drives data to the DUT through the signals / interface between itself and the DUT.

The sequence driver verifies the DUT by sending sequences to the BFM which drives them to the DUT to verify the functions of the MII. By default the sequence driver generates random predefined sequences. The sequences can be constrained to generate user-defined sequences to meet the verification environment requirements.

Most of the Ethernet PHY test scenarios will be implemented using sequences, to enhance the verification process, reduce simulation time, and base the verification on coverage-driven functional methodology and reusability.

The Ethernet PHY passive agent consists of:

- A monitor "ethernet_phy_monitor"
- A scoreboard "ethernet_phy_scoreboard".

The passive agent only monitors the DUT signals. The monitor collects the data from the DUT signals and trigger events on the status of traffic to and from the DUT. The monitor defines coverage points and checks that the DUT is working correctly with respect to the Ethernet Protocol.

The scoreboard verifies the integrity of data by comparing the send and received Ethernet packets.

The config blocks in the eVC are used to configure the eVC at the environment and the agent level.

The signal map blocks represent the signals interfaces of the agents.

4.3.3.2.2 Ethernet PHY Registers

In the Ethernet PHY eVC, registers are developed as constraints to configure the verification environment. These registers are taken form Intel LXT971A document [29]. Below is a list of the PHY registers used in the Ethernet PHY eVC.

- Phy Control Register

- MII Status Register
- Phy Identification Register1
- Phy Identification Register2
- Phy Configuration Register
- Phy Status Register
- Phy Interrupt Enable Register
- Phy Interrupt Status Register
- Phy LED Configuration Register

An example of the Phy registers "Phy Control Register" is shown in the diagram below".

Bit	Name	Description			Type [1]	Default
0.15	Reset	1 = PHY reset 0 = Normal operation			R/W SC	0
0.14	Loopback	1 = Enable loopback mode 0 = Disable loopback mode			R/W	0
0.13	Speed Selection	0.6	0.13	Speed Selected	R/W	Note 2
		1	1	Reserved		
		1	0	1000 Mbps (not supported)		
		0	1	100 Mbps		
		0	0	10 Mbps		
0.12	Auto-Negotiation Enable	1 = Enable auto-negotiation process 0 = Disable auto-negotiation process			R/W	Note 2
0.11	Power-Down	1 = Power-down 0 = Normal operation			R/W	0
0.10	Isolate	1 = Electrically isolate PHY from MII 0 = Normal operation			R/W	0
0.9	Restart Auto-Negotiation	1 = Restart auto-negotiation process 0 = Normal operation			R/W SC	0
0.8	Duplex Mode	1 = Full-duplex 0 = Half-duplex			R/W	Note 2
0.7	Collision Test	1 = Enable COL signal test 0 = Disable COL signal test			R/W	0
0.6	Speed Selection	0.6	0.13	Speed Selected	R/W	0
		1	1	Reserved		
		1	0	1000 Mbps (not supported)		
		0	1	100 Mbps		
		0	0	10 Mbps		
0.5:0	Reserved	Write as 0, ignore on Read			R/W	00000

1. R/W = Read/Write
   RO = Read Only
   SC = Self Clearing
2. Default value of Register bits 0.12, 0.13 and 0.8 are determined by the LED/CFG*n* pins (refer to Table 9 on page 30).

Table 1 : Control Register

4.3.3.2.3 Ethernet PHY eVC Features

The Ethernet PHY eVC consists of the following features:

- A set of configurable registers to set the functionality of the PHY eVC.
- A physical layer (cable) interface which receives serial data from the cable, assembles it to nibbles and sends it to the receive module (Ethernet MAC) together with the "data valid" marker.
- Sends small, medium, large, and bigger than standard Ethernet packets to MAC.
- Logic to send different Inter Frame Gaps (IFG is between 960ns for 100Mbps – 9600ns for 10Mbps.
- Logic to create 4bytes CRC and appends it to Ethernet packets.
- Logic to verify CRC received packets and flag errors when there is mismatches.
- Logic to add zero to seven bytes preamble and start frame delimiter (SFD) at the beginning of each packet transmission to MAC.
- Implements the Media Independent Interface (MII) as defined in the IEEE 802.3 standard.
- Logic to generate MII cRX_CLK and TX_Clk clocks (25Mhz for 100Mbps and 2.5Mhz for 10Mbps).
- Logic to create asynchronous Carrier Sense and Collision signals.
- Logic to create error signals.
- Logic to create Loopback traffic.
- Logic to create 10Mbps and 100Mbps (half and full speeds) operations.

4.3.3.2.4 Ethernet PHY Signals Interface

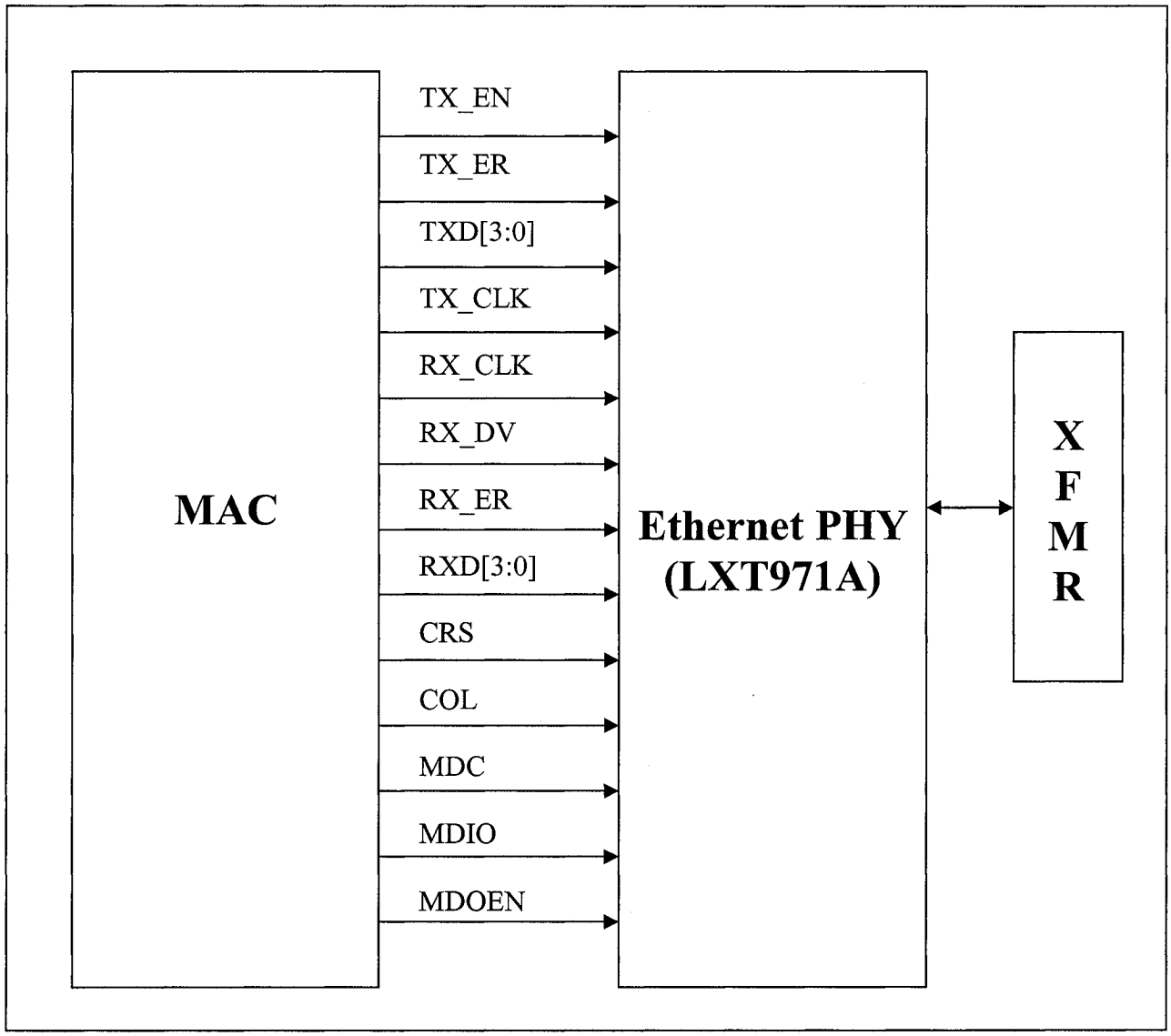

Figure 19: Ethernet PHY MII Interface

4.3.3.2.4.1 Ethernet PHY Signals Description

Symbol	Type	Signal Description
Data Interface Pins		
TXD3 TXD2 TXD1 TXD0	I	**Transmit Data**. TXD is a bundle of parallel data signals that are driven by the MAC. TXD<3:0> transitions synchronously with respect to the TX_CLK. TXD<0> is the least significant bit.
TX_EN	I	**Transmit Enable**. The MAC asserts this signal when it drives valid data on TXD. This signal must be synchronized to TX_CLK.
TX_CLK	O	**Transmit Clock**. TX_CLK is sourced by the PHY in both 10 and 100 Mbps operations. 2.5 MHz for 10 Mbps operation, 25 MHz for 100 Mbps operation.
RXD3 RXD2 RXD1 RXD0	O	**Receive Data**. RXD is a bundle of parallel signals that transition synchronously with respect to the RX_CLK. RXD<0> is the least significant bit.
RX_DV	O	**Receive Data Valid**. The LXT971A asserts this signal when it drives valid data on RXD. This output is synchronous to RX_CLK.
RX_ER	O	**Receive Error**. Signals a receive error condition has occurred. This output is synchronous to RX_CLK.
TX_ER	I	**Transmit Error**. Signals a transmit error condition. This signal must

		be synchronized to TX_CLK.
RX_CLK	O	**Receive Clock.** 25 MHz for 100 Mbps operation, 2.5 MHz for 10 Mbps operation.
COL	O	**Collision Detected.** The LXT971A asserts this output when a collision is detected. This output remains High for the duration of the collision. This signal is asynchronous and is inactive during fullduplex operation.
CRS	O	**Carrier Sense.** During half-duplex operation (Register bit 0.8 = 0), the LXT971A asserts this output when either transmitting or receiving data packets. During full-duplex operation (Register bit 0.8 = 1), CRS is asserted only during receive. CRS assertion is asynchronous with respect to RX_CLK. CRS is de-asserted on loss of carrier, synchronous to RX_CLK.
Type Column Coding: I = Input, O = Output		

Table 2: Ethernet PHY Signals Description

4.3.3.2.4.2 Ethernet PHY Signals Descriptions (Continued)

Symbol	Type	Signal Description
MII Control Interface Pins		
MDDIS	I	**Management Disable.** When

		MDDIS is High, the MDIO is disabled from read and write operations. When MDDIS is Low at power-up or reset, the Hardware Control Interface pins control only the initial or "default" values of their respective register bits. After the power-up/reset cycle is complete, bit control reverts to the MDIO serial channel.
MDC	I	**Management Data Clock**. Clock for the MDIO serial data channel. Maximum frequency is 8 MHz.
MDIO	I/O	**Management Data Input/Output**. Bidirectional serial data channel for PHY/STA communication.
MDINT	OD	**Management Data Interrupt**. When Register bit 18.1 = 1, an active Low output on this pin indicates status change. Interrupt is cleared by reading Register 19.
Type Column Coding: I = Input, O = Output, OD = Open Drain		
Network Interface Pins		
TPFOP TPFON	O	Twisted-Pair/Fiber Outputs, Positive & Negative. During 100BASE-TX or 10BASE-T operation, TPFOP/N pins drive 802.3 compliant pulses onto the line. During 100BASE-FX operation, TPFOP/N pins produce differential LVPECL outputs for fiber

		transceivers.
TPFIP TPFIN	I	Twisted-Pair/Fiber Inputs, Positive & Negative. During 100BASE-TX or 10BASE-T operation, TPFIP/N pins receive differential 100BASE-TX or 10BASE-T signals from the line. During 100BASE-FX operation, TPFIP/N pins receive differential LVPECL inputs from fiber transceivers.
SD/TP	I	**Signal Detect2**: Dual function input depending on the state of the device. **Reset and Power-Up.** Media mode selection: Tie High for FX mode (Register bit 16.0 = 1) Tie Low for TP mode (Register bit 16.0 = 0) **Normal Operation (FX Mode):** SD input from the fiber transceiver. **Normal Operation (TP Mode):** Tie to GND (uses an internal pulldown).
1. Type Column Coding: I = Input, O = Output, A = Analog, OD = Open Drain 2. For standard digital loopback testing (Register bit 0.14) in FX mode, the SD pin should be tied to an LVPECL logic High (2.4 V).		

Table 3: Ethernet PHY Signals Description (continued)

4.3.3.2.4.3 Ethernet PHY eVC Predefined Sequences

The Ethernet PHY eVC contains a sequence library with a set of predefined sequences. These sequences can be used to run different test scenarios on the Ethernet Mac IP Core design. Predefined sequences provide all the functionalities that are necessary to verify the DUT. Custom / user-defined sequences can be defined as well by extending existing sequences or defining new ones. Running sequences with different scenarios will eventually cover all the Functional points. Table 21 below lists the predefined Ethernet sequences.

Sequence Name	Description
PHY Registers	This sequence test all the PHY registers and set them to different values.
Reset	This is a reset sequence to test PHY reseting in different operational stages
100Mbps PHY Full-Duplex	This sequence will run 100Mbps PHY Full-Duplex operations
100Mbps PHY Half-Duplex	This sequence will run 100Mbps PHY Half-Duplex operations
10Mbps PHY Full-Duplex	This sequence will run 10Mbps PHY Full-Duplex operations
10Mbps PHY Half-Duplex	This sequence will run 10Mbps PHY Half-Duplex operations
PHY Frames Structures	This sequence will generate all types of PHY frames structures
PHY Collisions	This sequence will cover PHY collisions
PHY TX/RX Errors	This sequence will cover all the PHY TX/RX Errors
PHY State Machines	This sequence will test all the states of the PHY state machines
PHY Dribble bits	This sequence will generate PHY Dribble bits
PHY 4B/5B Coding	This sequence will generate random PHY 4B/5B coding
PHY LED Configuration	This sequence will test all the PHY LED Configuration settings
PHY Clock generation	This is to test the PHY Clock frequencies
Preambles	This sequence will handle all PHY preambles
PHY data	This sequence will generate random PHY data

Table 4: Ethernet PHY eVC Predefined Sequences

4.3.3.2.5 Ethernet PHY eVC Functional Coverage

Functional Coverage Item	Description
PHY Registers	This is to cover all the PHY registers
MII Data	This is to cover the data passes between the PHY LXT971A and Media Access Controller (MAC).
Management Data Configuration	This is to cover device configuration and management using the MDIO (Management Data IO) interface
Hardware Control Configuration	This to cover device configuration and management using the Hardware Control interface (using the LED driver to set device configuration)
MDIO Addressing	This is to cover the MDIO address ranges
Management Interface Read Frame Structure	This is to cover the different management Read frame data structure
MII Interrupts	This is to cover all the PHY MII Interrupts
MDIO Clock Speed	This is to cover the MDIO clock speeds
MDIO configuration register	This is to cover the line speed and operating conditions for the network link
Network Link Operation	This is to cover force network link operation settings, 100BASE-TX Full Duplex, 100BASE-TX Half-Duplex, 10BASE-T Full-Duplex, 10BASE-T Half-Duplex
Auto-negotiation	This is to cover the PHY LXT97A in auto-negotiation
Parallel detection	This is to cover the PHY LXT971A parallel detection
Software Power Down	This is to cover software power-down control (setting bit 11 in the PHY Control Register)
Reset	This is to cover the PHY reset operation
Transmit Enable	This is to cover TX_EN assertion / de-assertion
RX Data Valid	This is to cover the RX_DV timing, assertion, and de-assertion
Carrier Sense	This is to cover Carrier Sense operation (asynchronous, packet reception, Half-Duplex mode in transmission mode)
PHY Error signals	This is to cover all the PHY errors
collision	This is to cover PHY collisions (Half-Duplex line state, transmitter and receiver active at the same time)
PHY Loopback mode	This is to cover PHY loopback operation mode
10Mbps PHY Operation mode	This is to cover PHY 10Mbps operation modes

100Mbps PHY Operation mode	This is to cover PHY 100Mbps operation modes
PHY Frame Formats	This is to cover all the PHY data frames formats / structures
Transmission with Collisions	This is to cover PHY transmissions with collisions
Transmission without Collisions	This is to cover PHY transmissions without collisions
Reception with Errors	This is to cover PHY data reception with errors
Reception without Errors	This is to cover PHY data reception without errors
Preamble handling	This is to cover all the preamble handling with different preamble sizes and types
4Byte/5Byte Coding	This is to cover all kinds of 4B/5B coding
Link Failure	This is to cover PHY link failure
Dribble Bits	This is to cover PHY dribble bits in all modes
Link Integrity	This is to cover link integrity, link pulses transmission and etc.
LED functions	This is to cover all the PHY LED functions
LED Pulse Stretching	This is to cover the LED pulse stretching, 30, 60, and 100ms
PHY TX/RX state machines	This is to cover all the states of the PHY state machines

Table 5: Ethernet PHY eVC Functional Coverage

4.3.3.2.6 Checker points and rules

The PHY eVC checker will be implemented to test the PHY (MII) protocol signals, timings and the data integrity of TX/RX modules.

4.3.3.3 Wishbone eVC

4.3.3.3.1 Introduction

The Wishbone eVC is an e verification component designed to model the Wishbone Bus Interface Standard Protocol for the Ethernet Mac IP Core. The Wishbone Bus architecture is designed to be best fit for System-on-Chip Interconnection for semiconductor IP Cores. It offers a flexible integration solution that can be easily suited for a specific application. Its variety of bus cycles and data path widths solves today various system problems.

The Ethernet Mac IP Core is implemented with Master and Slave Wishbone bus interfaces to connect the core to the Host Bus which is using the Wishbone bus protocol. Master interface is used for storing the received data frames to the memory and loading the data that needs to be sent from the memory to the Ethernet core. Slave interface is used to access the Ethernet registers and buffer descriptors (BD) in the Ethernet Wishbone module.

The WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores is a flexible design methodology for use with semiconductor IP cores. Its purpose is to foster design reuse by alleviating System-on-Chip integration problems. This is accomplished by creating a common interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user. [51]

The WISHBONE interconnect is intended as a general purpose interface. As such, it defines the standard data exchange between IP core modules. It does not attempt to regulate the application-specific functions of the IP core.

The WISHBONE architects were strongly influenced by three factors. First, there was a need for a good, reliable System-on-Chip integration solution. Second, there was a need for a common interface specification to facilitate structured design methodologies on large project teams. Third, they were impressed by the traditional system integration solutions afforded by microcomputer buses such as PCI bus and VMEbus. [51]

The WISHBONE interconnection makes System-on-Chip and design reuse easy by creating a standard data exchange protocol. Features of this technology based on Wishbone Interconnection Architecture Specification are: [51]

- Makes System-on-Chip and design reuse easy by creating a standard data exchange protocol.
- Simple, compact, logical IP core hardware interfaces that require very few logic gates.
- Supports structured design methodologies used by large project teams.
- Full set of popular data transfer bus protocols including:

    - READ/WRITE cycle
    - BLOCK transfer cycle
    - RMW cycle
- Modular data bus widths and operand sizes.
- Supports both BIG ENDIAN and LITTLE ENDIAN data ordering.
- Variable core interconnection methods support point-to-point, shared bus, crossbar switch, and switched fabric interconnections.
- Handshaking protocol allows each IP core to throttle its data transfer speed.
- Supports single clock data transfers.
- Supports normal cycle termination, retry termination and termination due to error.
- Modular address widths.
- Partial address decoding scheme for SLAVEs. This facilitates high speed address decoding, uses less redundant logic and supports variable address sizing and interconnection means.
- User-defined tags. These are useful for applying information to an address bus, a data bus or a bus cycle. They are especially helpful when modifying a bus cycle to identify information such as:
    - Data transfers
    - Parity or error correction bits
    - Interrupt vectors
    - Cache control operations
- MASTER / SLAVE architecture for very flexible system designs.
- Multiprocessing (multi-MASTER) capabilities. This allows for a wide variety of System-on-Chip configurations.
- Arbitration methodology is defined by the end user (priority arbiter, round-robin arbiter, etc.).
- Supports various IP core interconnection means, including:
    - Point-to-point
    - Shared bus
    - Crossbar switch
    - Data flow interconnection
    - Off chip
- Synchronous design assures portability, simplicity and ease of use.
- Very simple, variable timing specification.
- Documentation standards simplify IP core reference manuals.
- Independent of hardware technology (FPGA, ASIC, etc.).
- Independent of delivery method (soft, firm or hard core).
- Independent of synthesis tool, router and layout tool technology.
- Independent of FPGA and ASIC test methodologies.
- Seamless design progression between FPGA prototypes and ASIC production chips.

4.3.3.3.2 Wishbone eVC Signals Interface

This section describes the signals interfaces between the MASTER interface, SLAVE interface, and SYSCON module. The figure below shows the Wishbone interfaces.

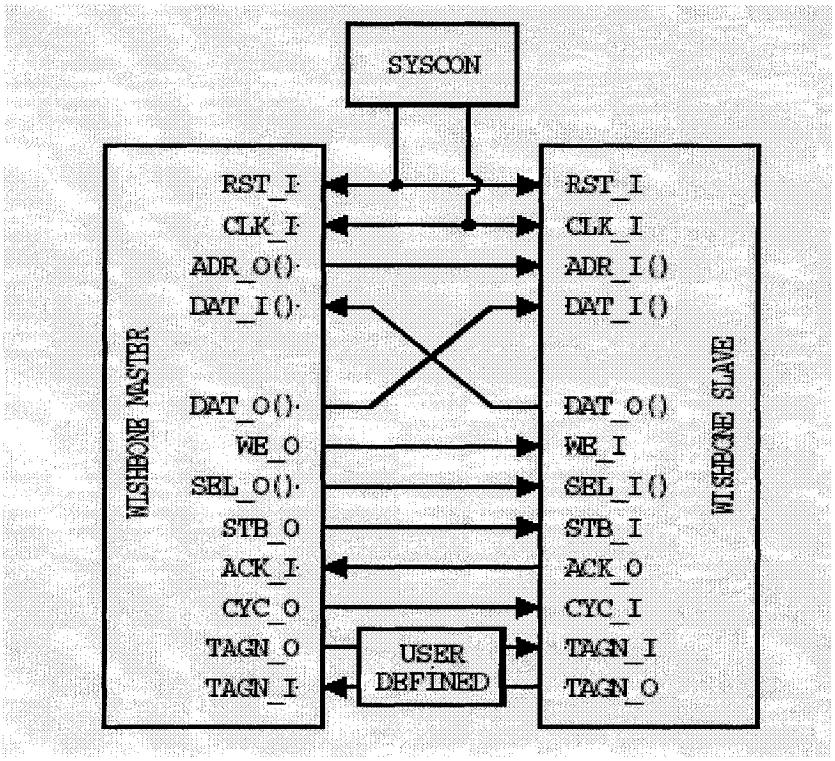

Figure 20: Wishbone Signals MASTER, SLAVE and SYSCON interfaces

4.3.3.3.2.1 Wishbone eVC Signals Description

This section describes the Wishbone's Signals. Some of these signals are optional and they don't have to be present on a specific interface using Wishbone as the Soc interconnection. This section is based on Wishbone SoC Architecture Specification document. [51]

4.3.3.3.2.1.1 SYSCON Module Signals

Signal Name	Description
CLK_O	The system clock output [CLK_O] is generated by the SYSCON module. It coordinates all activities for the internal logic within the WISHBONE interconnect. The INTERCON module connects the [CLK_O] output to the [CLK_I] input on MASTER and SLAVE interfaces.
RST_O	The reset output [RST_O] is generated by the SYSCON module. It forces all WISHBONE interfaces to restart. All internal self-starting state machines are forced into an initial state. The INTERCON connects the [RST_O] output to the [RST_I] input on MASTER and SLAVE interfaces.

Table 6: SYSCON Module Signals

4.3.3.3.2.1.2 Signals Common to MASTER and SLAVE Interfaces

Signal Name	Description
CLK_I	The clock input [CLK_I] coordinates all activities for the internal logic within the WISHBONE interconnect. All WISHBONE output signals are registered at the rising edge of [CLK_I]. All WISHBONE input signals are stable before the rising edge of [CLK_I].
DAT_I()	The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]).
DAT_O()	The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]).

RST_I	The reset input [RST_I] forces the WISHBONE interface to restart. Furthermore, all internal self-starting state machines will be forced into an initial state. This signal only resets the WISHBONE interface. It is not required to reset other parts of an IP core although it may be used that way.
TGD_I()	Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data input array [DAT_I()], and is qualified by signal [STB_I]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification.
TGD_O()	Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data output array [DAT_O()], and is qualified by signal [STB_O]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification.

Table 7: Signals Common to Master and Slave Interfaces

4.3.3.3.2.1.3 MASTER Signals

Signal Name	Description
ACK_I	The acknowledge input [ACK_I], when asserted, indicates the normal termination of a bus cycle.
ADR_O()	The address output array [ADR_O()] is used to pass a binary address. The higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size and granularity. For example the array size on a 32-bit data port with BYTE granularity is [ADR_O(n..2)]. In some cases (such as FIFO interfaces) the array may not be present on the interface.
CYC_O	The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_O] signal is asserted during the first data transfer, and remains asserted until the last data transfer. The [CYC_O] signal is useful for interfaces with multi-port interfaces (such as dual port memories). In these cases, the [CYC_O] signal requests use of a common bus from an arbiter.
ERR_I	The error input [ERR_I] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier. Also see the [ACK_I] and [RTY_I] signal descriptions.
LOCK_O	The lock output [LOCK_O] when asserted, indicates that the current bus cycle is uninterruptible. Lock is asserted to request complete ownership of the bus. Once the transfer has started, the INTERCON does not grant the bus to any other MASTER, until the current MASTER negates

	[LOCK_O] or [CYC_O].
RTY_I	The retry input [RTY_I] indicates that the interface is not ready to accept or send data, and that the cycle should be retried. When and how the cycle is retried is defined by the IP core supplier.
SEL_O()	The select output array [SEL_O()] indicates where valid data is expected on the [DAT_I()] signal array during READ cycles, and where it is placed on the [DAT_O()] signal array during WRITE cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_O(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port.
STB_O	The strobe output [STB_O] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_O()]. The SLAVE asserts either the [ACK_I], [ERR_I] or [RTY_I] signals in response to every assertion of the [STB_O] signal.
TGA_O()	Address tag type [TGA_O()] contains information associated with address lines [ADR_O()], and is qualified by signal [STB_O]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification.
TGC_O()	Cycle tag type [TGC_O()] contains information associated with bus cycles, and is qualified by signal [CYC_O]. For example, data transfer, interrupt acknowledge and cache control cycles can be uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE

	SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification.
WE_O	The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

Table 8: Master Signals

4.3.3.3.2.1.4 SLAVE Signals

Signal Name	Description
ACK_O	The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle.
ADR_I()	The address input array [ADR_I()] is used to pass a binary address. The higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size. For example the array size on a 32-bit data port with BYTE granularity is [ADR_O(n..2)].
CYC_I	The cycle input [CYC_I], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_I] signal is asserted during the first data transfer, and remains asserted until the last data transfer.
ERR_O	The error output [ERR_O] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier.
LOCK_I	The lock input [LOCK_I], when asserted, indicates that the current bus cycle is uninterruptible. A SLAVE that receives the LOCK [LOCK_I] signal is accessed by a single MASTER only, until either [LOCK_I] or [CYC_I] is negated.

RTY_O	The retry output [RTY_O] indicates that the interface is not ready to accept or send data, and that the cycle should be retried. When and how the cycle is retried is defined by the IP core supplier.
SEL_I()	The select input array [SEL_I()] indicates where valid data is placed on the [DAT_I()] signal array during WRITE cycles, and where it should be present on the [DAT_O()] signal array during READ cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_I(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port.
STB_I	The strobe input [STB_I], when asserted, indicates that the SLAVE is selected. A SLAVE shall respond to other WISHBONE signals only when this [STB_I] is asserted (except for the [RST_I] signal which should always be responded to). The SLAVE asserts either the [ACK_O], [ERR_O] or [RTY_O] signals in response to every assertion of the [STB_I] signal.
TGA_I	Address tag type [TGA_I()] contains information associated with address lines [ADR_I()], and is qualified by signal [STB_I]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification.
TGC_I()	Cycle tag type [TGC_I()] contains information associated with bus cycles, and is qualified by signal [CYC_I]. For example, data transfer, interrupt acknowledge and cache control cycles can

	be uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification.
WE_I	The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

Table 9: Slave Signals

4.3.3.3.3 Wishbone eVC Architecture

As it is shown in Figure 21: Wishbone eVC architecture, the Wishbone eVC consists of the following modules:

- Wishbone environment unit
- Wishbone active agent unit
- Wishbone passive agent unit
- Wishbone fields configuration unit
- Wishbone signals mapping unit
- Wishbone Transmit / Receiver sequence drivers and sequences
- Wishbone data sequence driver and sequences
- Wishbone Ethernet BFM
- Wishbone monitor / checker
- Wishbone functional coverage

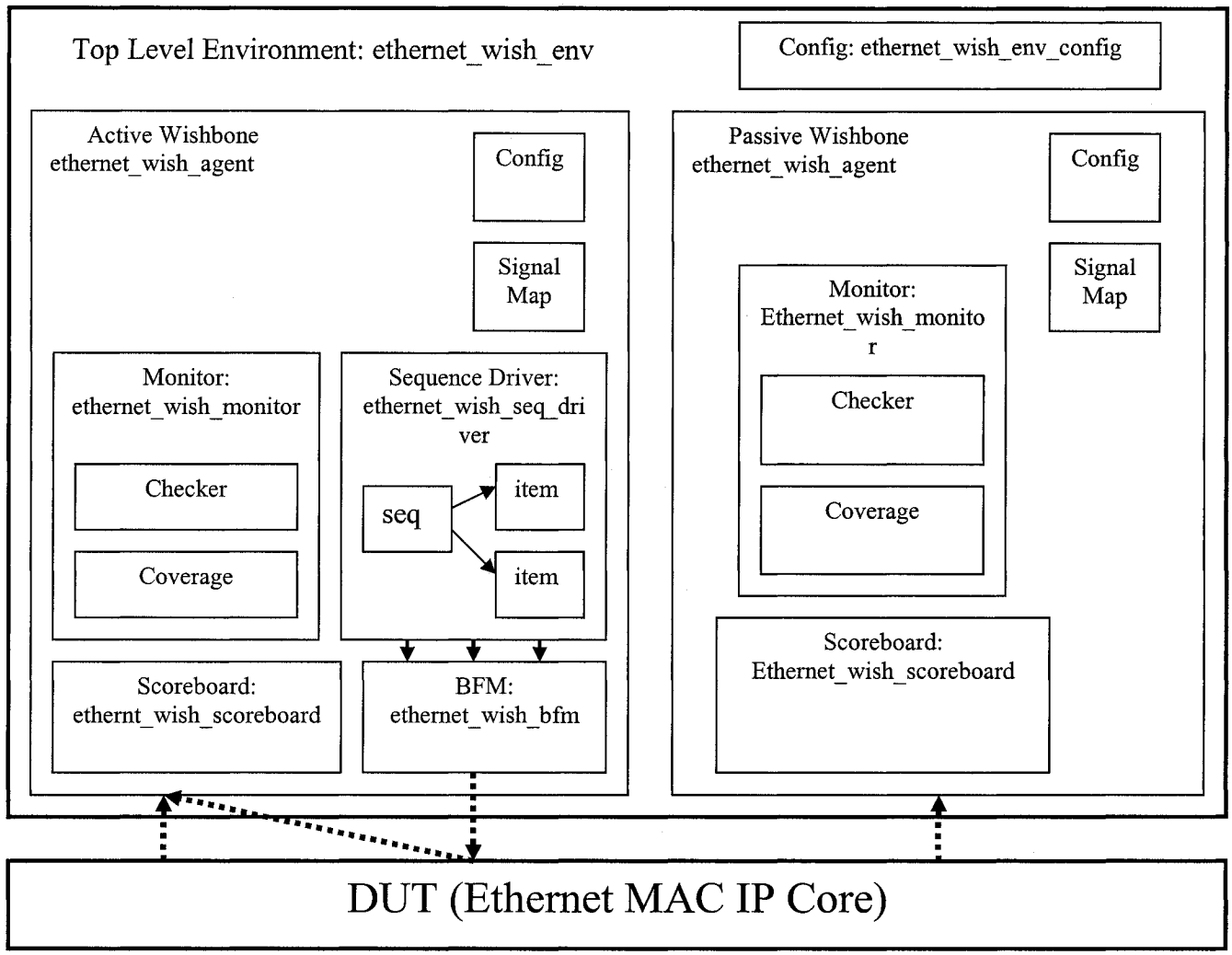

Figure 21: Ethernet Wishbone eVC Architecture

The Ethernet Wishbone eVC is represented by ethernet_wish_env.

The eVC simulates the behavior of the Wishbone Host Interface / Protocol.

The active Wishbone ethernet_wish_agent unit simulates the Wishbone Protocol behavior. The eVC verifies the DUT with the active Wishbone agent generating traffic to the DUT and responding to its traffic. The passive agent does not drive anything to DUT, however it passively monitors the bus signal interface between the DUT and Wishbone Host Interface model.

The Ethernet Wishbone active agent consists of:

- A sequence driver "ethernet_wish_seq_driver", and a BFM "ethernet_wish_bfm".
- A monitor "ethernet_wish_monitor" to do DUT's checking and collects functional coverage.
- A scoreboard "ethernet_wish_scoreboard" to compare the collected DUT's and expected data.

All the active transactions with the DUT are done by the BFM. The BFM is a driver which drives data to the DUT through the signals / interface between itself and the DUT.

The sequence driver verifies the DUT by sending sequences to the BFM which drives them to the DUT to verify the functions of the Ethernet IP Core. By default the sequence driver generates random predefined sequences. The sequences can be constrained to generate user-defined sequences to meet the verification environment requirements.

Most of the Ethernet Wishbone test scenarios will be implemented using sequences, to enhance the verification process, reduce simulation time, and base the verification on coverage-driven functional methodology and reusability.

The Ethernet Wishbone passive agent consists of:

- A monitor "ethernet_wish_monitor"
- A scoreboard "ethernet_wish_scoreboard".

The passive agent only monitors the DUT signals. The monitor collects the data from the DUT signals and trigger events on the status of traffic to and from the DUT. The monitor defines coverage points and checks that the DUT is working correctly with respect to the Wishbone Protocol.

The scoreboard verifies the integrity of data by comparing the send and received Ethernet packets.
The configuration blocks in the eVC are used to configure the eVC at the environment and the agent level.

The signal map blocks represent the signals interfaces of the agents.

4.3.3.3.4 Wishbone eVC Functional Coverage

The functional coverage for the Wishbone eVC is determined by the Functional Coverage points. The eVC will be built on random constrains where all the coverage points will be covered through running regressions with different seeds. Coverage groups are defined as monitor's events and inside these coverage groups, functional coverage items are defined. Custom coverage can be developed by extending the existing predefined coverage definitions. The verification will be considered complete when there is a 100% functional coverage collection. Table X below shows all the Wishbone functional coverage items, and their descriptions.

Functional Coverage Item	Description
reset	This is to make sure that the DUT was reseted in different simulation times (i.e. start, middle, and end of simulations).
counters	This is to check different counters values.
Single Read Transfer	Cover single read transfer
Single Write Transfer	Cover single write transfer
Block Read Transfer	Cover block read transfer
Block Write Transfer	Cover block write transfer
RMW Transfer	Cover RMW transfer
Host Packet Type	Cover Transmit and Receive frames
Host Packet Size	Cover packet's lengths
End Of Burst Transfer	Cover End of Burst transfer
Big ENDIAN Transfer	Cover Big Endian transfer
Little ENDIAN Transfer	Cover Little Endian transfer
Tag Types Transfers	Cover all Tag Types: parity, status, etc
Error Transfers	Cover packets with errors
Acknowledge Transfers	Cover packets ACKs transfers
Retry Transfers	Cover Retransmission transfers
Constant Address Burst Transfer	Cover constant Address burst transfers
Single Burst Transfer	Cover Burst single transfer
Synchronous and Asynchronous Transfers	Cover synchronous and asynchronous transfers
Wait States – Throttle Transfer	Cover transfers with throttling
Data Address ranges	Cover packets address ranges
Transfer cycles count	Cover transfer cycle counts
Buffer Descriptors sizes	Cover buffer descriptors RAM sizes
Rx Fifo sizes	Cover Receiver Fifo sizes
Tx Fifo sizes	Cover Transmitter Fifo sizes
Buffer Descriptors status	Cover BD status
Registers Accesses	Cover Ethernet registers accesses
Master Transmit State Machine	Cover states of Master TX state machine
Master Receive State Machine	Cover states of Master RX state machine
Word aligned addresses	Cover word aligned addresses
Word unaligned addresses	Cover word unaligned addresses
Buffer Descriptors addresses	Cover BD address ranges
Number of TX / RX Buffer Descriptors	Cover BD counter

Types of RAM accesses	Host, Transmitter, REceiver
TX / RX Fifos	Fifo_cnt, Fifo_depth, Fifo_data_width
Wishbone Protocol Errors	Cover wishbone protocol errors
Configuration Registers Accesses	Cover Ethernet configuration registers accesses
Memory Accesses	Cover Ethernet memory accesses
BD RAM depth	Cover BD Ram depth
BD RAM addresses	Cover BD Ram address ranges
Packets Padding	Cover frames padding
CRC Transfers	Cover CRC transfers
BD Collision	Cover BD collisions
BD Carrier Sense Lost	Cover BD Carrier Sense Lost transfer
Data Frame / Control Frame	Cover normal and control frames
Short Frame	Cover short frames transmission
Too Long Frame	Cover too long frames transmission
Frame Transmission	Cover frames transmission
Frame Reception	Cover frame reception
Phase Transfers	Cover phase transfers
Lock Transfers	Cover lock transfers

Table 10: Wishbone eVC Functional Coverage items

4.3.3.3.5 Wishbone eVC Checker's Points and Rules

The Wishbone eVC checker will check Protocol errors and Data integrity. The table below outlines all the checking points and rules.

reset	This is to check once Reset is asserted; all the modules within Wishbone module such as counter and state machines are reseted. All the modules should stay in the initialized state until the rising CLK edge that follows the negation of RST.
CYC_O duration	Master interfaces must assert CYC_O for the duration of Single Read/Write, Block, and RMW cycles. CYC_O MUST be asserted no later than the rising CLK_I edge that qualifies the assertion of STB_O. CYC_O MUST be negated no earlier than the rising

	CLK_I edge that qualifies the negation of STB_O. Master interfaces may assert CYC_O indefinitely.
CYC_I negation	SLAVE interfaces MAY NOT respond to any SLAVE signals when [CYC_I] is negated. However, SLAVE interfaces MUST always respond to SYSCON signals.
ACK_O, ERR_O, RTY_O	The cycle termination signals ACK_O, ERR_O, and RTY_O must be generated in responsemto the logical AND of CYC_I and STB_I.
Minimum Master / Slave interface signals	As a minimum, the MASTER interface MUST include the following signals: ACK_I, CLK_I, CYC_O, RST_I, and STB_O. As a minimum, the SLAVE interface MUST include the following signals: ACK_O, CLK_I, CYC_I, STB_I, and RST_I. All other signals are optional.
Slave ERR_O / RTY_O signals	If a SLAVE supports the ERR_O or RTY_O signals, then the SLAVE MUST NOT assert more than one of the following signals at any time: ACK_O, ERR_O or RTY_O.
STB_I	Slave's signals ACK_O, ERR_O, and RTY_O are asserted and negated in response to the assertion and negation of STB_I
Single Read Transfer	To check that Single Read Transfer complies with Wishbone Protocol rules.
Single Write Transfer	To check that Single Write Transfer complies with Wishbone Protocol rules.
Block Read Transfer	To check that Block Read Transfer complies with Wishbone Protocol rules.
Block Write Transfer	To check that Block Write Transfer complies with Wishbone Protocol rules.
RMW Transfer	To check that RMW Transfer complies with Wishbone Protocol rules.
End of Burst Transfer	End-Of-Burst indicates that the current cycle is the last of the current burst. The MASTER signals the slave that the burst ends after this transfer.
Constant Address Burst Cycle	A MASTER signaling a constant address burst MUST initiate another cycle, the next cycle MUST

	be the same operation (either read or write), the select lines SEL_O() MUST have the same value, and that the address array ADR_O() MUST have the same value.
Data Packets Frames Integrity	The checker will compare the output data with the expected / predicted data.

Table 11: Wishbone eVC Checker's Points and Rules

4.3.3.3.6 Wishbone eVC Sequences

The Wishbone eVC contains a sequence library with a set of predefined sequences. These sequences can be used to run different test scenarios on the Ethernet Mac IP Core. Predefined sequences provide all the functionalities that are necessary to verify the DUT. Custom / user-defined sequences can be defined as well by extending existing sequences or defining new ones. Running sequences with different scenarios will eventually cover all the Functional points. Table X below lists the predefined Ethernet sequences.

Sequence Name	**Description**
Wishbone Single Read Transfer	A sequence to test single read transfers
Wishbone Single Write Transfer	A sequence to test single write transfers
Wishbone Block Read Transfer	A sequence to test block read transfers
Wishbone Block Write Transfer	A sequence to test block write transfers
Wishbone RMW Transfer	A sequence to test RMW transfers
Wishbone Packets Types	A sequence to generate all the Wishbone packets' types
Wishbone Packets Lengths	A sequence to generate all the Wishbone packets' lengths
Wishbone Big Endian Transfer	A sequence to generate big endian transfers
Wishbone Little Endian Transfer	A sequence to generate little endian transfers
Wishbone Tags Transfers	A sequence to generate different valid Wishbone Tags transfers
Wishbone Errors Transfers	A sequence to generate Wishbone transfers with errors
Wishbone Waits Insertion Transfer	A sequence to generate Wishbone transfers with waits insertions (Throttling)
Wishbone Address Ranges	A sequence to generate Wishbone transfers with

	different address ranges
Wishbone counters ranges	A sequence to generate Wishbone transfers to test all the counters' ranges
Wishbone Tx Fifo	A sequence to generate Wishbone transfers to test Tx fifo depths and etc.
Wishbone RX Fifo	A sequence to generate Wishbone transfers to test Rx Fifo depths and etc.
Wishbone Protocol Errors	A sequence to generate Wishbone transfers with protocol errors
Wishbone Buffer Descriptors depths	A sequence to generate transfers to test Wishbone Buffer Descriptors depths
Wishbone Buffer Descriptors Addresses	A sequence to generate transfers for different Buffer Descriptors addresses
Wishbone Data Frames	A sequence to generate different valid Wishbone data frames
Wishbone Control Frames	A sequence to generate control frames
Wishbone Phase Transfers	A sequence to generate phase transfers
Wishbone Lock Transfers	A sequence to generate lock transfers
Wishbone Packets with Padding	A sequence to generate packets with different padding lengths
Wishbone Registers	A sequence to generate valid transactions to test all the Wishbone registers
Wishbone CRC	A sequence to generate transfers with valid and invalid CRC

Table 12: Wishbone eVC Predefined Sequences

4.3.3.4 Ethernet Mac eVC

The Ethernet Mac eVC will be very simple and passive in actions. Since most of the work will be done by the Ethernet PHY eVC and Wishbone Host eVC, the Ethernet Mac will be mainly focusing on collecting functional coverage and reporting any holes in testing the Ethernet Mac IP Core.

4.3.3.4.1 Ethernet Mac eVC Architecture

As it is shown in Figure 22: Ethernet Mac eVC architecture, the Ethernet Mac eVC consists of the following modules:

- Ethernet Mac environment unit
- Ethernet Mac passive agent unit

- Ethernet Mac fields configuration unit
- Ethernet Mac signals mapping unit
- Ethernet Mac monitor / checker
- Ethernet Mac functional coverage

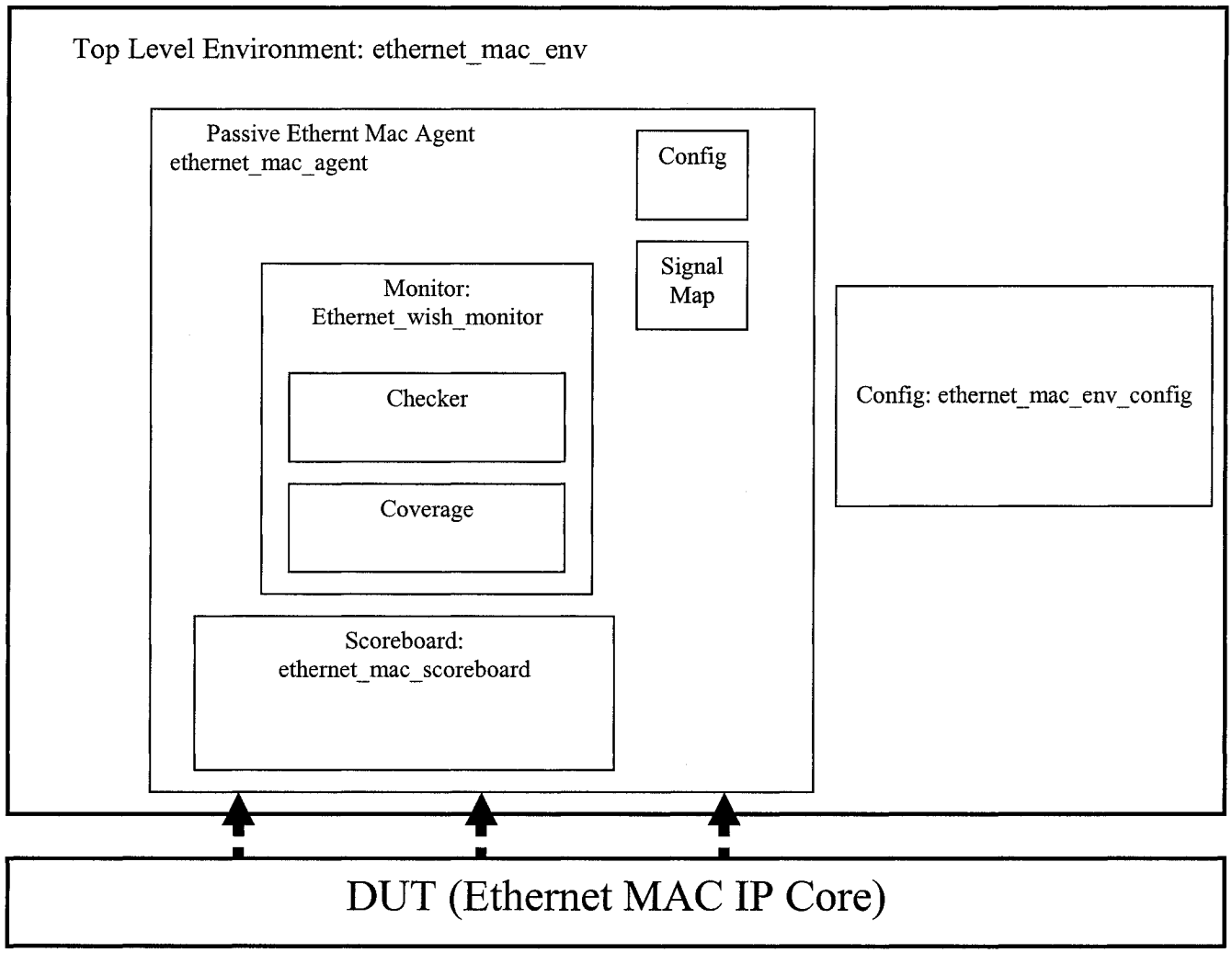

Figure 22: Ethernet Mac eVC Architecture

The Ethernet Mac eVC is represented by ethernet_mac_env.

The Ethernet Mac eVC behaves as a passive entity to monitor all the Ethernet transfers and report the amount of functions covered for testing the Ethernet Mac IP Core. The main component of the Ethernet Mac eVC is the passive agent which focuses on monitoring the Ethernet traffic and collect functional coverage.

The Ethernet Mac passive agent consists of:

- A monitor "ethernet_mac_monitor"
- A scoreboard "ethernet_mac_scoreboard".

The passive agent only monitors the DUT signals. The monitor collects the data from the DUT signals and trigger events on the status of traffic to and from the DUT. The monitor defines coverage points and checks that the DUT is working correctly with respect to the Ethernet Mac Protocol.

The scoreboard verifies the integrity of data by comparing the send and received Ethernet packets.

The configuration blocks in the eVC are used to configure the eVC at the environment and the agent level.

The signal map blocks represent the signals interface of the passive agent.

4.3.3.4.2 Ethernet Mac eVC Functional Coverage

Functional Coverage Item	Description
Preamble Lengths	Cover all the Ethernet packets' preamble lengths such as 32-bit preamble
Full duplex	Cover full duplex transactions
Half duplex	Cover half duplex transactions
Ethernet Registers	Cover all the Ethernet registers
PHY Registers	Cover all the Ethernet PHY configuration registers
MII Bit counter	Cover the primary counter for the MII interface
Ethernet PHY Address	Cover PHY addresses. Several PHY chips might be connected to the MII interface.
MII Command Register	Cover the commands written to the MII Command registers.
PHY Read Status	Cover PHY read status transfers such as LinkFail status
PHY Scan Status	Cover PHY Scan status transfers
Preamble Removal	Cover preamble removals
Ethernet Rx counters	Cover Ethernet receive counters such as byte counter, IFG counter, and Delay CRC counter
Ethernet RX state machines	Cover Ethernet receive state machine states such as Idle, Drop, Preamble, SFD, Data0, and Data1 states
Broadcast Packets	Cover Ethernet broadcast packets
Multicast Packets	Cover Ethernet multicast packets
Ethernet Receiver Operations	Cover Ethernet receiver operations such as HugEn (Reception of big packets), DlyCrcEn (Delayed CRC check), IFG

	(Minimum Inter Frame Gap), and etc.
Ethernet Packets lengths	Cover all the Ethernet packets lengths
RX packet abortion	Cover drops of Ethernet RX packets
Start of data frame marker	Cover the start of data frame pointer
End of data frame marker	Cover the end of data frame pointer
Ethernet TX status signals	Cover Ethernet transmitter status signals such as done, retry, abort ,error, and etc
Ethernet collision delay	Cover Ethernet random delay when back off is performed after collisions
Ethernet TX counter	Cover Ethernet transmission counters such as Delay CRC counter, Nibble Counter, Byte counter, and etc.
Ethernet TX state machine	Cover Ethernet transmission state machines states such as Idle, Preamble, Data0, Data1, PAD, FCS, IPG, Jam, BackOff and Defer states
Ethernet TX Abort	Cover all the scenarios of the Ethernet TX abort scenarios such as Packet is too long, Wishbone underrun, excessive deferral state, late collision, maximum number of collisions
Ethernet TX signals	Cover Ethernet transmission signals such as WillTransmit, CollisionReset, CollisionWidnow, RetryCounter, DataCrc, CrcEnable, InitializeCrc and etc.
Collision Jam pattern	Cover the Ethernet jam pattern.
Ethernet Data frame length	Cover all the Ethernet data frame lengths such as data length >= min. frame length, data length < min. frame length, data with padding, and etc.
Ethernet Pause Control Frames	Cover Ethernet pause control frames
Ethernet Errors	Cover Ethernet errors such as RX Error, Rx CRC Error, Rx invalid symbol, and etc.
Ethernet Buffer Descriptors status	Cover the Ethernet BD status such as TX/RX errors, RX late collision, RX short frames, RX Big frames, RX Dribble Nibble, TX Retry count, TX Retry Limit, TX late collision, TX Defer, TX Carrier sense lost, TX underrun, RX overrun, and RX miss
Ethernet Wishbone buffer descriptors	Cover the Ethernet Wishbone buffer descriptors
Ethernet Wishbone RX / TX Fifos	Cover the Ethernet Wishbone RX / TX Fifos such as fifo width, fifo depth, fifo data width, fifo full, fifo empty and etc.
Ethernet Memory Addresses	Cover the Ethernet Wishbone memory addresses
Ethernet Buffer Descriptors addresses	Cover the Ethernet BD addresses

Ethernet PHY clock frequencies	Cover the Ethernet MII clock dividers and fequencies
Ethernet PHY registers	Cover the Ethernet PHY registers such as registers values, addresses (correct and wrong addresses) and etc
Ethernet PHY status / command signals	Cover the Ethernet PHY status signals such as stop scan command after read/ write requests, busy / nvalid status during write / scan, linkfail, and etc.
Ethernet Data Shift	Cover the Ethernet data shift registers
Ethernet TX Packet's Fields	Cover the Ethernet TX packet fields
Ethernet RX Packet's Fields	Cover the Ethernet RX packet fields
Ethernet TX Packet Kind	Cover the Ethernet TX packet kind
Ethernet RX Packet Kind	Cover the Ethernet RX packet kind
Ethernet RX Packet Error Kind	Cover the Ethernet RX packet error kind
Ethernet TX Packet Error Kind	Cover the Ethernet TX packet error kind
Ethernet Protocol Error Kind	Cover the Ethernet protocol error kind
Ethernet Tag kinds	Cover all the Ethernet tag kinds
Ethernet Packets with alignment errors	Cover Ethernet packet's alignment errors
Ethernet CRC error length	Cover Ethernet CRC error length
Ethernet CRC error timing	Cover Ethernet CRC error timing
Ethernet Data error length	Cover Ethernet Data error length
Ethernet Timing error	Cover Ethernet Timing error

Table 13: Ethernet MAC eVC Functional Coverage

4.3.3.4.3 Ethernet Mac eVC Checker points and rules

Checker point	**Description**
Good CRC	Check that packets are transmitted with correct / valid CRC
Good SFD	Check that packets with correct Start of Frame Delimiters are transmitted
Jumbo packets	Check the size of the jump packets
Short packets	Check the size of the minimum packets

Packets with valid pause	Check the valid pause opcode
Packet address alignment	Check packets address alignments
Packets with valid preambles	Check packets preamble length
Half duplex mode – pause frame	Check the pause frame in half duplex mode
Packets with unicast source address	Check that the PHY sends packets with unicast source address
Tagged frames	Check the correctness of frames tags
Check packet padding length	Check that the padding length is correct
Minimum Inter Frame Gap	Check the min. IFG between frames transmissions in Half duplex mode
Mac 1st data nibble	Check the value of the 1st Mac data nibble preamble to the PHY
PHY Carrier Sense	Check that the PHY asserts CRS when it transmits / receive packets to / from the MAC
PHY 1st data nibble	Check the value of the 1st PHY data nibble
TX enable and Collision Delay	Check the delay between the TX enable and Collision rise / fall
False Carrier	Check the False Carrier indication with RX_Error / RX_DV signals
TX enable and Collision occurrence	Check the TX_EN is low when the collision signal is de-asserted
False Carrier	Check the False Carrier indication with RX_Error / RX_DV signals
TX enable and Collision occurrence	Check the TX_EN is low when the collision signal is de-asserted
Isolation Mode	Check the isolation mode
Carrier Sense Assertion	Check that the CRS signal remains asserted throughout the collision assertion.
Carrier Sense De-assertion	Check that the CRS signal remains de-asserted when PHY is not transmitting or receiving data in Half Duplex Mode

Table 14: Ethernet Mac eVC Checker's Points and Rules

4.3.3.4.4 Ethernet Mac eVC Sequences

The Ethernet eVC doesn't have a sequence driver to drive the Ethernet Mac IP Core. Since it only consists of a passive agent, it will monitor all the activities in the Ethernet Mac IP Core (DUT) and collect functional coverage accordingly. However when Ethernet eVC is hooked up with Wishbone eVC and Ethernet PHY eVC under an Ethernet Top verification environment, certain sequences will be generated using both Wishbone and PHY eVC sequence drivers. Table 34 below lists some of the main predefined Ethernet sequences.

Sequence Name	**Description**
Normal Ethernet Packets	This sequence run normal Ethernet packets without errors based on IEEE 802.3 standard
Ethernet Packets Errors	This sequence generates Ethernet packets with different errors.
Ethernet Packets No Errors	This sequence generates Ethernet packets with random kinds and with no errors.
Ethernet Packets Random Length	This sequence generates legal Ethernet packets with random lengths
Ethernet Packets Padding	This sequences generates packets with data length less than 46 bytes where padding is required
Ethernet Packets CRC	This sequence generates Ethernet packets with correct and corrupted CRC
Ethernet Packets No Padding	This sequence generates Ethernet packets with small data length and no padding
Ethernet Packets SFD	This sequence generate packets with valid and invalid SFD
Ethernet Packets Pause Opcode	This sequence generate packets with valid and invalid pause opcode
Ethernet Packets Drop	This sequence generates packets with drops

Table 15: Ethernet Mac eVC Predefined Sequences

4.3.3.4.5 Ethernet Top Verification Environment

The Ethernet Top Level Verification Environment is the container which consists of different eVC blocks to build a unified testbench (Verification Environment) to verify thoroughly the Ethernet Mac IP Core

(DUT). As it is shown in Figure 23 "Ethernet Top Level Verification Environment", the Ethernet Top Level Verification Environment consists of:

- Ethernet PHY eVC
- Wishbone Host eVC
- Ethernet Mac eVC
- Configuration unit
- Ethernet Top Verification unit

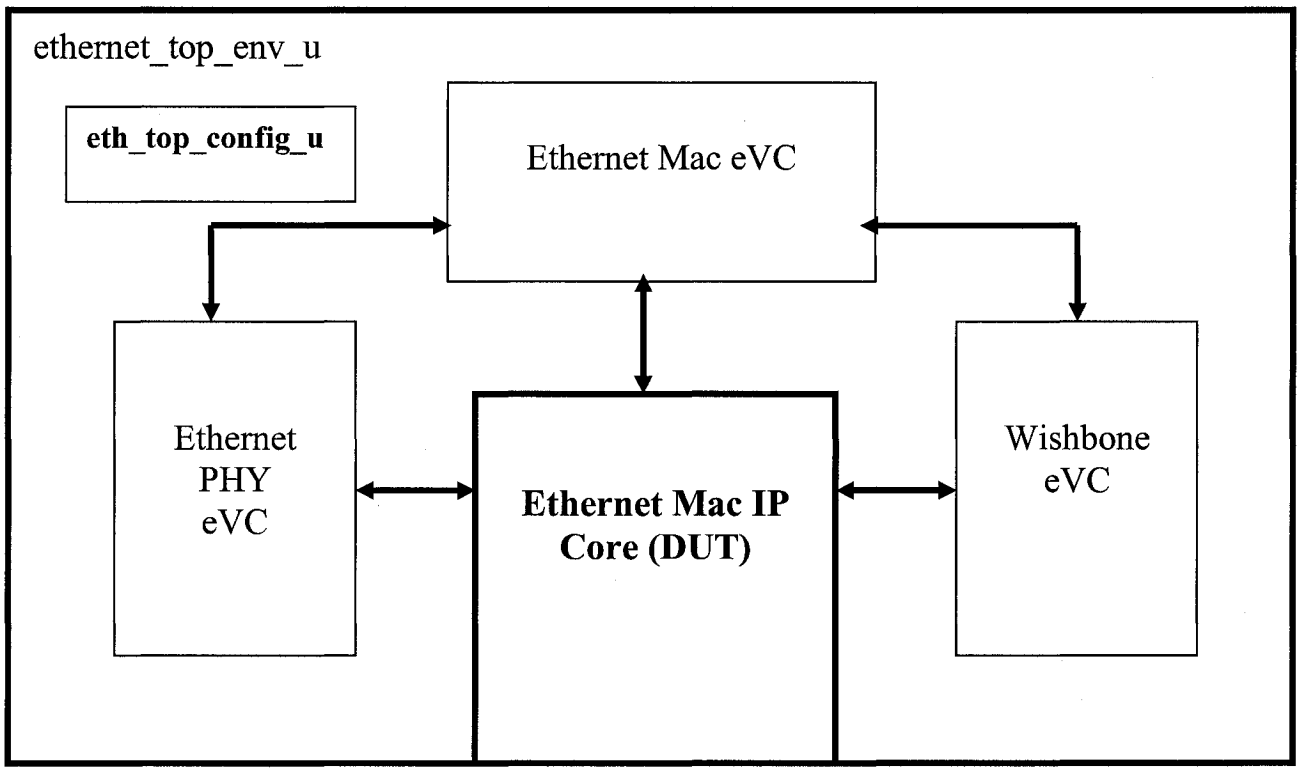

Figure 23: Ethernet Mac Top Level Verification Environment

Based on the functional coverage results, new test scenarios (directed, semi-directed) will be developed by building new user-defined / custom Ethernet sequence to cover any functional holes of the Ethernet Mac IP Core design (DUT).

Chapter 5

5 Ethernet Verification Environment Algorithms and Comparison

5.1 Ethernet Mac IP Core Deterministic Testbench Algorithm

The Ethernet Mac IP Core Deterministic Testbench Algorithm is a directed testbnech and based only on one file called "tb_ethernet.v". This Verilog file contains everything. All the tests are within this file. The tests are written as Verilog tasks. Once the testbench is compiled an executable file is created. To run the simulation, only the name of the executable file is used without any options. Two reference models are used to simulate both the HOST Interface and Ethernet PHY interface. Both reference models are developed in a behavioral Verilog and Verilog tasks are used to force certain scenario. The Ethernet Mac IP Core Deterministic Testbench doesn't have a regression suite since the testbench consists of only one executable file resulting from compiling the tb_ethernet.v. The whole testbench is built around directed testcases. No randomness is applied since it is constrained by the power of the Verilog language where functional coverage and randomness are not part of its features. Finally the testbench is not built on reusability where it will be hard to maintain and impossible to reuse for feature verification efforts (i.e. it will be very hard to reuse the testbench to verifiy other Ethernet designs. Diagram X below shows the structure and blocks of the Ethernet Mac IP Core Deterministic Testbench.

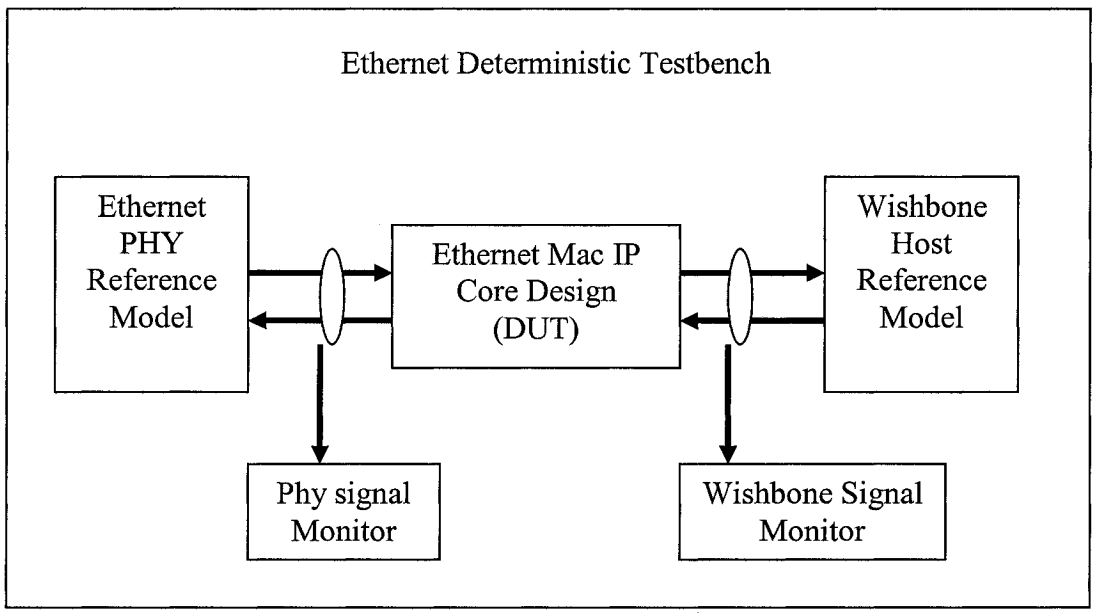

Figure 24: Ethernet Mac IP Core Deterministic testbench

5.2 *Coverage-Driven Functional Verification and Reuse Methodology Algorithm*

The Coverage-Driven Functional Verification and Reuse Methodology Algorithm is based completely on functional coverage through randomness and sequences. The testbench is built on reusability where libraries of sequences are generated to collect 100% functional coverage and to able to reuse them for future designs. The testbench is very modular and based on objected oriented programming. The testbench is built on a plug and play concept where modules are used to built the whole verification environment. The testbench consists of eVCs e Verification Components where each verification component can be used as a whole verification environment (testbench) or part of a system verification environment where multiple eVCs are used to build the whole system testbench. In verifying the Ethernet Mac IP Core, a system level testbench is built. The testbench consists of 3 eVCs: Ethernet PHY eVC, Wishbone eVC, and Ethernet Mac eVC. The system level testbench is called etherent_mac_ip_core_verification. The etherent_mac_ip_core_verification is the top level environment which used the 3 above eVCs to build a complete verification environment to verify the Ethernet Mac IP Core Design (DUT). eRMs (e Ruse Methodology) is used where sequences are generated to test all the functionalities of the design. Each eVC consists of agents, monitors, bfms, checkers, scoreboards, sequence drivers and sequence items. The Ethernet PHY eVC is built to simulate the Intel Ethernet PHY LXT971A. The Wishbone eVC simulate a Host interface (CPU) for the Ethernet Mac IP Core (i.e. it contains memory and it programs Ethernet Mac IP Core registers). The Ethernet Mac eVC consists of a slave agent to monitor the Ethernet Mac IP Core interfaces and to collect functional coverage. The testbench consists of scripts to run regression suites. Different scenarios and functional coverage points are collected by running the testbench multiple times with random seeds. During regressions if a test failed, the same test will be run with the same seed, however this time waveforms are generated / dumped. All these regressions are manageable through scripts developed in the testbench. Diagram X below shows the structure and algorithm of the Coverage-Driven Functional Verification and Reuse Methodology.

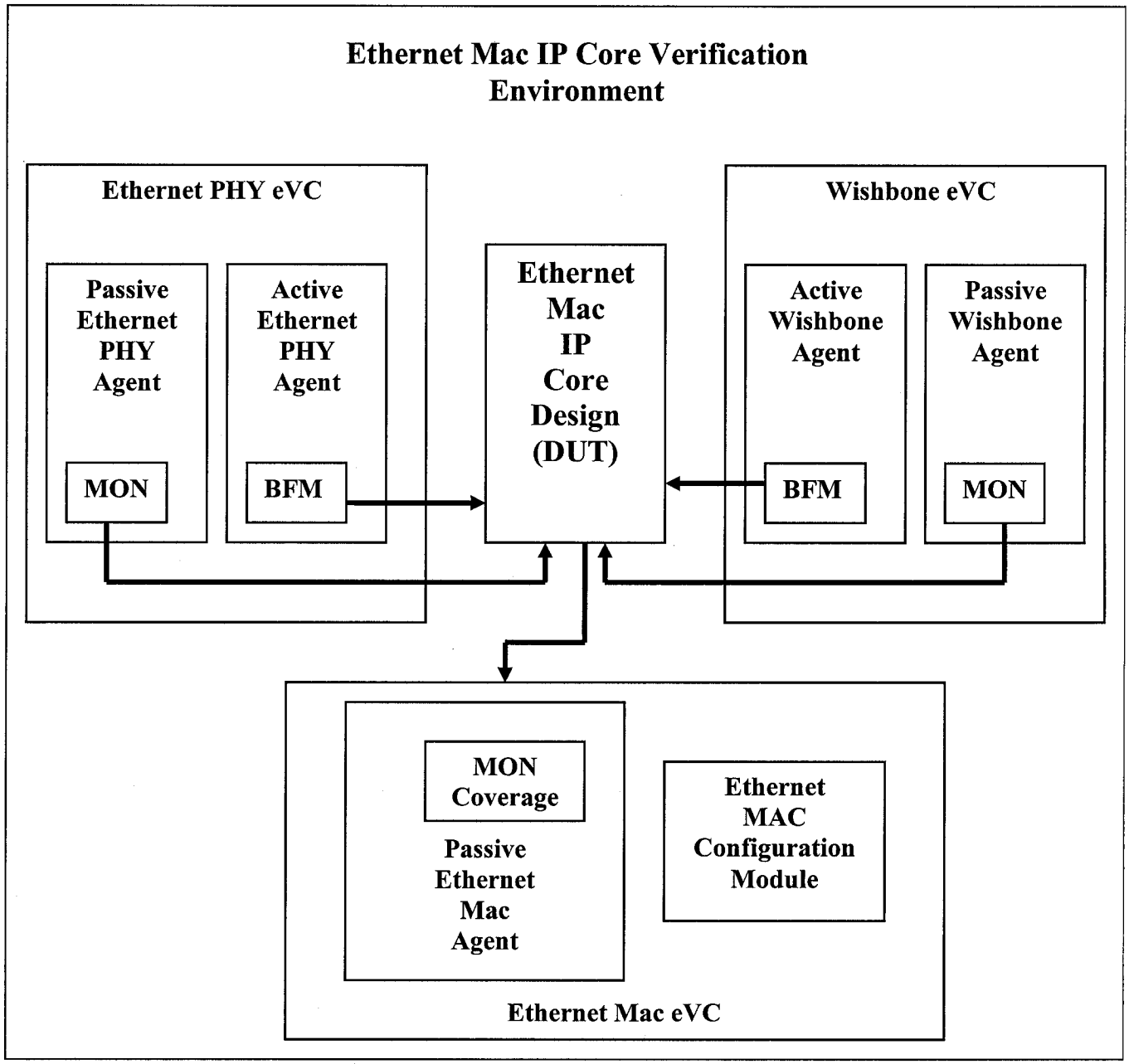

Figure 25: Coverage-Driven Functional Verification and Reuse Methodology Structure and Algorithm

5.3 *Ethernet Deterministic Testbench vs. Coverage-Driven Functional Verification and Reuse Methodology Testbench*

1)

The Ethernet Mac IP Core testbench is a deterministic testbench where only directed tests are written and run against the design (DUT). Deterministic testbench only covers a limited scope of the design since it is limited by the number of testcases written and what the verification engineer thought of at the time of writing the directed testcases. Hence there is no clear indicator when the verification process will be complete. More bugs could be found by running more testcases.

The Coverage-Driven Functional Verification Methodology is based on randomness and pure functional coverage. With minimum number of sequence running with different seeds (random data / stimulus generation) all the functions of the design (DUT) will be covered. Once there are no holes in the functional coverage (i.e. all the functions of the design are complete), the verification process considered complete. Since there is a 100% functional coverage collection, no more bugs will be in the design by running more random sequences.

2)

The Ethernet Mac IP Core testbench is not built on an object oriented programming where modules and reusability are applied. It only consists of a one large Verilog file (1000s of Verilog code lines) "tb_ethernet.v" to test the Ethernet Mac IP Core Design. It would be very difficult to maintain and re-use the testbench for future designs.

The Coverage-Driven Functional Verification and Reuse Methodology is built on reusability and object oriented programming. It is built on the concepts of plug and play where unit and system level verification environments are built by using eVCs (e Verification Components and their sub-modules). Since it is built on eRM (e Reuse Methodology), it is easy to re-use and maintain.

3)

The Ethernet Mac IP Core testbench doesn't have the concept of regression. You only need to run the tb_ethernet file where verilog tasks for different Ethernet scenarios are written. To run the same testbench with different values, the tb_ethernet testbench needs to be modified manually. So it doesn't matter how many times you run the tb_etherent, you still get the same results except you modified the testbench "tb_ethernet.v" manually.

The Coverage-Driven Functional Verification and Reuse Methodology is built on the concepts of regression where the same testbench is run again and again with random seeds. Basically in the Specman command line or script, the vcs_specman executable is run with seed=random option where each time the test runs different data values and scenarios are injected and tested against the Ethernet Mac IP Core Design (DUT).

4)

The Ethernet Mac IP Core testbench is not a protected verification environment. It is not built on the concepts of inheritance and encapsulation. People tend to open the testbench verilog files and modify them without any kind of control. There isn't a well-defined test writer interface where the verification environment is encapsulated from tests writers.

The Coverage-Driven Functional Verification and Reuse Methodology is built on inheritance and encapsulation. A well-defined test writer interface is outlined for engineers to develop tests without knowing the details and structure of the verification environment. Doing that we simplify the life of the test writers and we protect the verification environment from any changes. Verification engineers who developed the verification environment are the only one who can modify and update the verification environment since they develop it and they know the ins and outs of it. Test writers don't need even to know the Specman language to write tests. With very short time (i.e. within hours) they can contribute in writing sequences to test their RTL designs. These sequences are simply a set of constrains.

5)

Running the Ethernet Mac IP Core testbench, few errors were found.

Running the Coverage-Driven Functional Verification and Reuse Methodology environment many more errors were found especially to cover all the Ethernet Mac IP Core functionalities.

6)

It is proven through previous verification experiences that a deterministic verification environment lacks efficiency and credibility. Thousands of testcases are required to verify only directed scenarios. Many engineers and overhead work are required to develop all the directed testcases. Many hi-tech companies such as Intel, Cisco System, Alcatel and much more use very advanced verification languages such as Specman to verify their multi-million ASICs. They hardly develop any deterministic testbenches. From previous engineering working experiences, the deterministic testbench approach was always avoided and

advanced languages such as Specman and Vera were used to verify complex designs based on Coverage-Driven functional verification and reusability. Time to market and reusability are crucial criteria to the success of any hi-tech company.

7)

Using the Ethernet Mac IP Core Deterministic testbench, most likely bugs will turn out late in the development cycle of the design since a deterministic testbench approach was used. Having bugs late in the design cycle is very costly and projects are slipped.

Using the Coverage-Driven Functional Verification and Reuse Methodology testbench, rarely bugs will turn out in late in the design cycle. Verification was based on design's functions. The verification was considered complete when all the functions are covered without holes. Having a well-defined and advanced verification approach as that will keep peace in mind, meet timelines, and enhance success and credibility.

8)

Using the Ethernet Mac IP Core Deterministic testbench, the runtime is large and inefficient. It takes few hours to run the testbench. This is very small design. What will happen if this approach is used on a chip level design where multiple modules are used? It will take to run the testbench. This is very costly and it will never be used in industry where time and money is the factor for the success of any hi-tech company. Engineers' time are vary valuable and expensive (costly), so time is very important factor in design development cycle.

Using the Coverage-Driven Functional Verification and Reuse Methodology testbench, runtime is very fast and efficient. Within few minutes a test scenario is complete. This verification methodology is very ideal for chip and system level verification. Most of the hi-tech companies nowadays use these advanced verification tools to verify their complex multi-million gates designs. Their development doesn't survive or continue without having well-defined verification methodology as Specman. Verification consumes more than 70% of any design development cycle. It is a must to have an advanced verification methodology and tool. Without it no (fabless) semiconductor company can survive in developing and selling their designs / products.

9)

Using the Ethernet Mac IP Core Deterministic testbench, there are no features provided for dumping waveforms, outputting reports or having any debugging mechanism except the $display system task in

Verilog to print messages into the screen. Since verification is a crucial part and the longest in design development cycle, a verification tool is needed to provide all the necessary features to develop verification environment's modules and debugging their models and designs. The Ethernet Mac IP Core Deterministic Testbench only provides the verilog language to write the Ethernet testbench. It is well known that Verilog can not be used as a verification tool and language to verify complex designs. So using Verilog for verification in this Ethernet design is an ultimate direction to failure.

Using the Coverage-Driven Functional Verification and Reuse Methodology testbench, lots of features are provided for verification engineers to develop efficiently their design verification environment. Messaging mechanism (Enabling, Disabling, LOW, MEDIUM, HIGH and lots more), Reports (Functional Reports), Debugging and Error Tracing mechanism (step into RTL, Specman Code, break points and etc.), and regression suites tools. These are only some of the main feature of an advanced verification language as Specman (Cadence). Lots of additional features are embedded in the tool to facilitate the work of verification engineers to ultimately produce ASIC designs free of bugs.

10)

Using the Ethernet Mac IP Core Deterministic testbench, the cost of doing verification is almost free. Basically it doesn't cost money to use Verilog. Even lots of free simulators are out there which can be downloaded for free. Having said that, this doesn't permit or explain the reasons behind using Verilog as the verification tool to verify ASIC designs. In fact doing that will cost the company millions of dollars or cost them the closure of their business since they will produce chips / designs with bugs. They will lose their customers / business forever.

Using the Coverage-Driven Functional Verification and Reuse Methodology testbench, the verification tool is costly. Semiconductor Companies usually need few licenses for their engineers and this might cost them 10s of thousands. There is no other solution except doing that. It is a must to produce RTL designs without bugs. So there is no alternative for using an advanced and a well-defined verification methodology and tool. In fact using these verification tools is an eventual success for companies since they will produce designs free of bugs and build customers trust and credibility.

11)

Using the Ethernet Mac IP Core Deterministic testbench, no clear test plan and design features to be tested are outlined in the testbench document. Only few scenarios are provided especially regarding the MII (Media Independent Interface). Also no separate checkers or scoreboards are provided.

Using the Coverage-Driven Functional Verification and Reuse Methodology testbench, clear test scenarios, functions to be tests, checking points and etc. are outlined clearly. Separate monitors, checkers and scoreboard are developed. Going over the functional coverage points and the eVCs included in verifying the RTL design, would give a clear picture what have been tested and how modules are correlated together.

12)

The understandability / readability of the Ethernet Mac IP Core Deterministic Testbench is not trivial and easy. There is a need to go over / read the tb_ethernet.v file in order to understand what the testbench is doing and how the Verilog tasks are working. It is very difficult to keep up with the 1000s of Verilog code lines.

It is very easy to understand the Coverage-Driven Functional Verification and Reuse Methodology environment since it is built around collecting the functions of the RTL design and the fact that it is very modular. By looking at the Functional Coverage points and the checking items, anybody could understand clearly what the ASIC design (DUT) is doing.

Chapter 6

6 Conclusion

6.1 Summary

In this thesis, we tried to solve today ASIC verification problem. Nowadays ASICs are getting large and very complex. This imposes a difficult task on the verification effort. The verification task is around 70% of the ASICs Designs life Cycle (Development). Hence a new verification approach is required to replace all the traditional verification approaches to reduce cost, to increase profitability / credibility, to enhance the verification process, and to meet the market demands on time. We chose Coverage-Driven Functional Verification and Reuse Methodology as the verification solution for today multi-millions ASICs Designs. We used Specman Elite e language as the verification tool since it has all the features for coverage-driven functional verification and reuse methodology. Its random generator engine, functional coverage capabilities and reports, and reusability make it the leading verification tool and language. In this thesis , We provided detailed Test Plan and design verification environment based on CDV and RM Methodology. Both OpenCores' Deterministic Verilog Testbench and Coverage-Driven Functional Verification and Reuse Methodology results were shown and a fair comparison was outlined. We proved that Coverage-Driven Functional Verification and Reuse Methodology is the right solution for today's ASICs verification crisis.

6.2 Recommendations

Our recommendation for today ASIC verification crisis is to use a well defined verification methodology with full system automation. Nowadays the competition is very strong among chip design companies. Time to market is very crucial and hence companies are forced to deliver their products quickly even though their designs are not completely verified. To be able to do quick and dirty verification, hardware and verification engineers are forced to write limited directed test cases to their ASIC designs. As a result of that many ASIC re-spins are required and millions of dollars are spent. Eventually if companies don't make profit or don't break even, they are forced to lay off engineers and eventually the whole economy gets affected. So companies need to adopt and force the usage of well defined verification methodologies as Coverage-Driven Functional Verification and Reuse Mythology to organize their verification tasks and succeed in producing free of bugs ASIC designs.

6.3 Future Work

In our Coverage-Driven Functional Verification and Reuse Methodology Environment, we built a simple Ethernet PHY reference model based on Intel 971LX and OpenCores PHY Verilog Model. We were forced to use a simple PHY reference model to have a fair comparison with OpenCores Deterministic testbench. For future work, to verify thoroughly the Ethernet MAC IP Core, a full and detailed Ethernet PHY reference model is required to verify all the features and functionalities of the Ethernet MAC IP Core. Also we could build reference models using systemC language. Most of hi tech-companies use systemC for reference models since it is powerful in developing cycle accurate and transaction models. SystemC gets integrated easily with Specman e language and hence it enforces the Coverage-Driven and Functional Verification and Reuse Methodology by enhancing the checking mechanisms. Also for future work, we can add assertions to the design to help us in pin pointing the cause of failures. Assertions are good to have features to maximum the force of the Verification task. However for this thesis it was not required since we developed monitors, checkers and scoreboards to verify data and protocol integrity.

Bibliography

[1] A. Aharon, D. Goodman, M. Levinger, Y. Lichtenstein, Y. Malka, C. Metzger, M. Molcho & G. Shurek, "Test Program Generation for Functional Verification of PowerPC Processors in IBM", Proceedings of the Design Automation Conference, pp.279-285, 1995.

[2] P.-H. Ho et al., "Smart Simulation Using Collaborative Formal and Simulation Engines," *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design, Digest of Technical Papers*, IEEE Press, Piscataway, N.J., 2000, pp. 120-126.

[3] T. B. Alexander, K. A. Dickey, D. N. Goldberg, R. V. La Fetra, J. R. McGee, N. Noordeen, and A. Prakash. *Verification, characterization, and debugging of the HP PA 7200 processor.* In *Hewlett-Packard Journal*, pages 1–12, February 1996.

[4] D. P. Appenzeller and A. Kuehlman, "Formal Verification of a PowerPC Microprocessor", Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers & Processors, pp 79-84, IEEE, October 1995.

[5] L. Arditi and G. Clave, "A Semi-formal Methodology for the Functional Validation of an Industrial DSP System," *Proc. IEEE Int'l Symp. Circuits and Systems*, IEEE Press, Piscataway, N.J., 2000, pp. 205-208.

[6] G.Barrett, M.Belhadj, C.Berthet, A.McIsaac and F.Rocheteau, "The Application of Design Abstraction and Transistor Abstraction in an Industrial Design Flow", submitted to FMCAD, 1996.

[7] Lionel Bening and Harry Foster, "Principles of Verification RTL Design", Kluwer Academic Publisher, 2000.

[8] M. Benjamin et al., "A Study in Coverage-Driven Test Generation," *Proc. 36th Design Automation Conf.*, ACM Press, New York, 1999, pp. 970-975.

[9] Janick Bergeron, *"Writing Testbenches, Functional Verification of HDL Models"*, Kluwer Academic Publishers, 2000.

[10] R.E. Bryant, "Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis. In Proceedings of the International Conference on Computer-Aided Design, pages 350-353, 1991.

[11] D.V. Campenhout, T. Mudge, and J.P Hayes, "High-Level Test Generation for Design Verification of Pipelined Microprocessors," *Proc. 36th Design Automation Conf.*, ACM Press, New York, 1999, pp. 185-188.

[12] F. Casaubieilh, A. McIsaac, M. Benjamin, M. Bartley, F. Pogodalla, F. Rocheteau, M. Belhadj, J. Eggleton, G. Mas, G. Barrett, C. Berthet, "Functional Verification Methodology of Chameleon Processor", Chameleon Programme, SGS-THOMSON Microelectronics.

[13] S. Devadas, A. Ghosh, and K. Keutzer, "An Observability-Based Code Coverage Metric for Functional Simulation," *Proc. 33rd Design Automation Conf.*, ACM Press, New York, 1996, pp. 418-425.

[14] A.Th.Eiriksson and K.L.McMillan, "Using Formal Verfication / Analysis Methods on the Critical Path in System Design: A Case Study", in P.Wolper (ed.), CAV'95, Lecture Notes in Computer Science 939, Springer Verlag, 1995.

[15] F. Fallah, S. Devadas, and K. Keutzer, "OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Simulation," *Proc. 35th Design Automation Conf.*, ACM Press, New York, 1998, pp. 152-157.

[16] M. Faltas, "Verisity – Verification Process Automation (VPA)", Verifiy2003: Verification Process Automation from ESL to Gates, October 30th, 2003.

[17] L. Fournier, A. Koyfman, and M. Levinger, "Developing an Architecture Validation Suite: Application to the PowerPC Architecture," *Proc. 36th Design Automation Conf.*, ACM Press, New York, 1999, pp. 189-194.

[18] D. Geist, M. Farkas, A. Lander, Y. Lichtenstein, S. Ur, and Y. Wolfstal, "Coverage-Directed Test Generation Using Symbolic Techniques". FMCAD, November 1996.

[19] A. Ghosh, S. Devadas and A.R. Newton, "Sequential Logic Testing and Verification", Kluwer Academic Publishers, 1992.

[20] R. Grinwald et al., "User Defined Coverage - A Tool Supported Methodology for Design Verification," *Proc. 35th Design Automation Conf.*, ACM Press, New York, 1998, pp. 158-163.

[21] I. Gronau, A. Hartman, A. Kirshin, K. Nagin, S. Olvovsky, "A Methodology and Architecture for Automated Software Testing",

http://www.haifa.il.ibm.com/projects/verification/gtcb/papers/gtcbmanda.pdf

[22] A. Gupta, S. Malik, and P. Ashar, "Toward Formalizing a Validation Methodology Using Simulation Coverage," *Proc. 34th Design Automation Conf.*, ACM Press, New York, 1997, pp. 740-745.

[23] R.C. Ho and M.A. Horowitz, "Validation Coverage Analysis for Complex Digital Designs," *Proc. Int'l* Formal Verification 10 **IEEE Design & Test of Computers**
Conf. Computer-Aided Design, ACM Press, New York, 1996, pp. 322-325.

[24] C.N. Ip, "Simulation Coverage Enhancement Using Test Stimulus Transformation," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design, Digest of Technical Papers*, IEEE Press, Piscataway, N.J., 2000, pp. 127-133.

[25] M. Kantrowitz and L.M. Noack, "I'm Done Simulating; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha Microprocessor," *Proc. 33rd Design Automation Conf.*, ACM Press, New York, 1996, pp. 325-330.

[26] T. Kuhn, T. Oppold, C. Schulz-key, M. Winterholer, W. Rosenstiel, M. Edwards, Y. Kashai, "Object Oriented Hardware Synthesis and Verification", seminar paper, available online at http://www.inf.ethz.ch/personal/julah/seminar_papers/hw_synth%20ws%202004-2005/1/p1.pdf

[27] Barakatain Leila, Tahar Sofiene, Lamarche Jean, Gendrean Jean-Marc, "Functional Verification of a SCI-PHY Level 2 Protocol Engine, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada.

[28] D. Lewin, D. Lorez, S. Ur "A Methodology for Processor Implementation Verification", FMCAD, November 1996.

[29] Intel LXT971A 3.3V Dual-Speed Fast Ethernet PHY Transceiver Datasheet. Available online at http://www.kip.uni-heidelberg.de/ti/DCS-Board/current/datasheets/ethernet/LXT971A.pdf

[30] Y. Malka, A. Ziv, "Design Reliability – Estimation Through Statistical Analysis of Bug Discovery Data", in Proceedings of the 35th Design Automation Conference, New York, pages 644-649, June 1998.

[31] C. Malley & M. Dieudonne, "Logic Verification Methodology for PowerPC Microprocessors", In Proceedings of the Design Automation Conference, pp. 234-240, 1995.

[32] Igor Mohor, "Ethernet IP Core Design Document", revision 0.4, 29/10/2002, available online at http://www.opencores.com/cgi-bin/cvsget.cgi/ethernet

[33] Igor Mohor, "Ethernet IP Core Product Brief", revision 0.4, 29/10/2002, available online at http://www.opencores.com/cgi-bin/cvsget.cgi/ethernet

[34] Igor Mohor, "Ethernet IP Core Specification", revision 0.4, 27/10/2002, available online at http://www.opencores.com/cgi-bin/cvsget.cgi/ethernet

[35] D. Moundanos, J.A. Abraham, and Y.V. Hoskote, "A Unified Framework for Design Validation and Manufacturing Test," *Proc. Int'l Test Conf.*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 875-884.

[36] D. Moundanos, J.A. Abraham, and Y.V. Hoskote, "Abstraction Techniques for Validation Coverage Analysis and Test Generation," *IEEE Trans. Computers*, vol. 47, no. 1, Jan. 1998, pp. 2-13.

[37] G. Nativ, S. Mittermaier, S. Ur, A. Ziv, **IBM** Corporation, "Cost Evaluation of Coverage Directed Test Generation for the IBM Mainframe", http://www.research.ibm.com/pics/verification/ps/FeedbackCDG_itc2001

[38] Samir Palnitkar, *"Design Verification with e"*, Pearson Education, Inc, 2004

[39] Mishra Prabhat, Dutt Nikil, "Functional Coverage Driven Test Generation for Validation of Pipelined Processors", CECS Tchnical Report #04-05, Center for Embedded Computer Systems, University of California, Irvine, CA 92697, CA, March 12th, 2004. Available online at http://www.cecs.uci.edu/technical_report/TR04-05.pdf

[40] M. Puig-Medina, G. Ezer, and P. Konas, "Verification of Configurable Processor Cores," *Proc. 37th Design Automation Conf.*, ACM Press, New York, 2000, pp. 426-431.
Theory and Application, vol. 16, nos. 1-2, Feb. 1999, pp. 67-81.

[41] J. Shen and J.A. Abraham, "An RTL Abstraction Technique for Processor Microarchitecture Validation and Test Generation," *J. Electronic Testing:*
Theory and Application, vol. 16, nos. 1-2, Feb. 1999, pp. 67-81.

[42] J. Shen and J.A. Abraham, "Verification of Processor Microarchitectures," *Proc. 17th IEEE VLSI Test Symp.*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 189-194.

[43] S. Tasiran, K. Keutzer, "Coverage Metrics for Functional Validation of Hardware Designs", IEEE Design & Test of Computers, Vol. 18 I. 4, July-Aug 2002.

[44] S. Taylor et al., "Functional Verification of a Multiple-Issue, Out-of-Order, Superscalar Alpha Processor—the DEC Alpha 21264 Microprocessor," *Proc. 35th Design Automation Conf.*, ACM Press, New York, 1998, pp. 638-643.

[45] S. Ur and Y. Yadin, "Micro Architecture Coverage Directed Generation of Test Program," *Proc. 36th Design Automation Conf.*, ACM Press, New York, 1999, pp. 175-180.

[46] S. Ur and A. Ziv, "Off-the-Shelf vs. Custom Made Coverage Models, Which Is the One for You?" *Proc. Software Testing, Analysis, and Review* (STAR 98), CD-ROM, Software Quality Engineering, Orange Park, Fla., 1998.

[47] R. Vemuri and R. Kalyanaraman, "Generation of Design Verification Tests from Behavioral VHDL Programs Using Path Enumeration and Constraint Programming," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 2, June 1995, pp. 201-214.

[48] Verisity Design Inc., white paper "Coverage-Driven Functional Verification", 2005. Available online at http://www.verisity.com/resources/whitepaper/coverage_driven.html

[49] Verisity Design Inc., white papter, "e Verification Components", 2005, available online at http://www.verisity.com/products/evc.html

[50] B. Wile, M. P. Mullen, C. Hanson, D. G. Bair, K. M. Lasko, P. J. Duffy, E. Kaminski, Jr., T. E. Gilbert, S. M. Licker, R. G. Sheldon, W. D. Wollyung, W. J. Lewis, R. J. Adkins, "Functional Verification of the CMOS S/390 Parallel Enterprise Server G4 System," IBM J. Res. Develop. 41, No. 4/5, 549-566, July/September 1997.

[51] WISHBONE SoC Architecture Specification, Revision B.3, available online at http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf

[52] H. Zhu, P.V. Hall, and J.R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol. 29, no. 4, Dec. 1997, pp. 366-427.