VERIFICATION OF ROW VOLTAGE CONTROL BLOCK IN THE NAND FLASH MEMORY

A Thesis submitted in partial fulfillment of the requirement for the Award of the Degree of

MASTER OF TECHNOLOGY

in VLSI Design

Submitted By

JYOTI THAKUR

Roll No. 601762009

Under Supervision of

Dr. Urvinder Singh

Designation- Associate Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB
JULY, 2019

CERTIFICATE

Western Digital®

SanDisk India Device Design Centre Pvt. Ltd.
Survey No.143/1, Amani Bellandur Khane Village
Prestige Excelsior, Prestige Tech Park
Marathalli-Sarjapur Outer Ring Road, Kadubeesanahalli
Varthur Hobli, Bengaluru-560103, India
Phone: +91-80-42422000
Telefax: +91-80-42422900
CIN: U72300KA2005PTC035961
www.sandisk.com

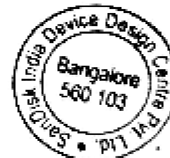
12 July 2019

To Whomsoever It May Concern

This is to certify that **Jyoti Thakur**, student of **Thapar Institute of Engineering & Technology, Patiala(Punjab)** has interned with, **SVA Team, Bangalore** during the period from Aug-09-2018 to July-12-2019 under the partial fulfillment of the **M.Tech Postgraduate Degree in VLSI.**

During this period, **Jyoti** was involved in the project **'Functional Verification Of Row Voltage Control Block'** inside NAND flash memory, under the guidance of **Atul Kulshreshtha, Manager, STM team, Western Digital.**

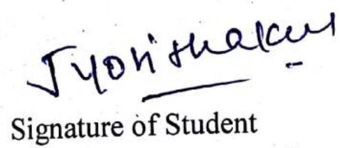
Thank you.



Surinder Bhagat
Senior Director– Human Resources

DECLARATION

I, **Jyoti Thakur** hereby declare that the work presented in this project " **Verification of Row Voltage Control Block In the NAND Flash Memory** "in completely fulfillment of the requirement for the award of degree of **Master of Technology (VLSI Design)** submitted at **Electronics and Communication department**, Thapar Institute of Engineering & Technology (Deemed to be University), Patiala is an authentic record of work carried out under supervision of **Dr. Urvinder Singh (Associate Professor, ECED, TIET)** from August 2018 to July 2019.The matter presented in this has not been submitted either in part or full to any other university or institute for the award of any other degree.



Signature of Student

Dr Urvinder Singh
Associate Professor
ECED TIET (Patiala)

ACKNOWLEDGMENT

The completion of the task could not be completed without acknowledging those persons whose guidance and continued support made our efforts a successful one.

I take this opportunity to express my profound gratitude and deep regard to my guide Dr. Urvinder Singh, faculty, Department of Electronics and Communication Engineering, Thapar Institute of Engineering & Technology for his guidance and constant encouragement throughout the course of this project.

I express my deep gratitude and indebtedness to Mr. VIJAY CHINCHOLE (Sr. Manager, SVA Team), Mr. ATUL KULSHRESHTHA (Manager, SVA Team) and Mrs. SNEHA SRISHYLAM (Sr. Engineer) at SanDisk, a Western Digital Brand, Bangalore for providing guidance and valuable suggestions with constant support throughout the project. I consider myself very fortunate to work with them. In addition, I would like to thank the whole team for their valuable feedback during my project work.

I extend my sincere thanks to Dr. ALPANA AGARWAL, Head of the Department of Electronics and Communication Engineering, Thapar Institute of Engineering & Technology ,Patiala ,for her co-operation, rendering all kinds of supports throughout the project work.

I also like to acknowledge the multitude of endeavors done by my friends for their kindest co-operation in bringing this work to a successful completion.

ABSTRACT

The correctness of a digital circuit is an important consideration in the design of digital systems. Given the extremely high and increasing costs of manufacturing microchips, the consequences of flaws getting unnoticed in system designs until after the production phase, would be very expensive. At the same time, RTL verification is still one the most challenging activities in digital system development. Different methodologies such as UVM are now majorly used for verification because of the reusability of the components in verification environment.

In this project work, verification of **"Row Voltage Control"** module inside NAND Flash memory is done. Testing is said to have completed once 100% coverage is attained. Functional coverage model is plugged to DUT along with UVM test bench and checked for 100% coverage in **"Row Voltage Control"**block.

TABLE OF CONTENTS

CHAPTER 1 10

INTRODUCTION 10

1.1 FLASH MEMORY: THEORY AND APPLICATIONS 10

1.1.1 NOR Memories 11

1.1.2 NAND Memories 11

1.1.3 Difference between NAND and NOR Flash memories 11

1.2 INTRODUCTION TO NAND FLASH MEMORY 12

1.2.1 Principle of operation of a NAND Flash Cell 12

1.2.2 Floating-gate Transistor 12

1.2.3 NAND Flash Cells are basically of two types 13

1. Single Level Cell (SLC): 13

2. Multi Level Cell (MLC): 13

1.3 NAND FLASH OPERATIONS 15

1.3.1 Program/Write 15

1.3.2 Read operation 16

1.3.3 Erase operation 16

1.4 VERIFICATION METHODOLOGIES USED IN INDUSTRY 17

1.5 TEST BENCH PRINCIPLES 18

1.6 TEST BENCH ARCHITECTURE 19

1.7 COVERAGE CLOSURE 20

1.8 FUNCTIONAL COVERAGE 20

1.8.1 Cover points 21

1.8.2 Cross coverage 21

1.9 FORMAL VERIFICATION 22

1.9.1 INTRODUCTION: 22

1.9.2 FORMAL EQUIVALENCE CHECKING: ... 23
1.9.3 FORMAL PROPERTY CHECKING: ... 23
1.9.4 HOW FORMAL VERIFICATION? ... 23
1.9.5 DRAWBACK OF SIMULATION FOR FUNCTIONAL VERIFICATION: ... 24
1.9.6 ADVANTAGE OF FORMAL OVER SIMULATION: ... 24
1.9.7 CONS OF FORMAL VERIFICATION ... 25
1.10 OBJECTIVE OF THE PROJECT ... 25
1.11 THESIS ORGANISATION OF REPORT ... 25
CHAPTER 2 ... 26
LITERATURE SURVEY ... 26
2.1 STATEMENT OF PROBLEM BASED ON IDENTIFIED RESEARCH GAPS ... 28
CHAPTER 3 ... 30
VERIFICATION OF MODULES INSIDE NAND FLASH MEMORY ... 30
3.1 ROW VOLTAGE CONTROL BLOCK ... 30
3.2 VERIFICATION FLOW CHART ... 31
3.3 VERIFICATION STEPS ... 33
CHAPTER 4 ... 34
IMPLEMENTATION OF VERIFICATION ... 34
METHODOLOGY AND RESULTS ... 34
4.1 COVERAGE MODEL ... 34
4.1.2 Functional coverage model ... 36
4.1.3 Unique cover points: ... 37
4.1.4 Uncovered Report ... 37
4.1.5 Test cases ... 38
4.2 RESULTS ... 38
4.2.1 Comparison of coverage percentage ... 39
CHAPTER 5 ... 40
FORMAL VERIFICATION OF OPERATION MODULE ... 40
5.1 INTRODUCTION ... 40

5.2 NEED FOR COMMAND MODULE 40
5.3 ARCHITECTURE AND ASSERTIONS 41
5.3.1 FLAVOR1 & FLAVOR2 42
5.3.1.1 ASSERTIONS FOR CHECKING SIGNAL ASSERTION 42
5.3.1.2 ASSERTIONS FOR CHECKING SIGNAL DE-ASSERTION 42
5.3.2.1 ASSERTIONS FOR CHECKING SIGNAL ASSERTION 43
5.3.2.2 ASSERTIONS FOR CHECKING SIGNAL DE-ASSERTION 43
5.3.3 ASSERTIONS TO CHECK ASYNC RESET 44
5.4 SPECIFICATION DOCUMENT 44
5.5 NEED FOR AUTOMATION 45
5.6 FORMAL VERIFICATION 45
5.8 REASONS FOR ASSERTION FAILURE 47
5.9 CONCLUSION: 47
CHAPTER 6 48
CONCLUSION 48
BIBLIOGRAPHY 49

LIST OF FIGURE

Figure 1 Floating Gate Transistor [1] 12
Figure 2 Write operation of NAND flash cell [1] 15
Figure 3 Read mode operation of NAND flash cell [1] 16
Figure 4 Erase operation of NAND flash cell [1] 17
Figure 5Test bench-design environment [2] 19
Figure 6Test bench Architecture[2] 19
Figure 7 Row Voltage Control Block 30
Figure 8 Executable Specification Format 31
Figure 9 Verification Flow Chart 32
Figure 10 Coverage Automation Flow 35
Figure 11 Input Executable File 35
Figure 12 Functional coverage model 36
Figure 13 Unique cover point 37
Figure 14 Uncovered Report 37
Figure 15 Test cases 38
Figure 16(i) Coverage results for erase before regression 38
Figure 17(ii) Coverage results for erase after regression 39
Figure 18(i) Coverage results for Program and Read before regression 39
Figure 19 Verification flow diagram 41
Figure 20Specification Format 44
Figure 21Results before Debug 46
Figure 22Results after Debug 46

CHAPTER 1

INTRODUCTION

Verification of digital designs plays a very important role in present day chips because of the increase in complexity of the design. It is pointed out that almost 60% of the entire chip design time is devoted to verification. A reasonable confidence on the functionality of the circuit can be acquired by verification of RTL design, assuming that manufacturing fault is not present. Before moving to the expensive chip manufacturing, all possible errors in the design must be removed. This is the underlying motivation of the verification. Whenever the functional errors are found, to reflect the proper behavior model is modified. Verification team originate all possibilities of tests and pass it to the design in the verification environment to check whether the design behaves as per the given specifications. In case, there is any failure, the functional design model needs to be modified to provide the true behavior of design given by specification and the RTL design updated accordingly.

New methodology for verification like OVM and UVM are gaining popularity because of the support extended by the major EDA tool vendors and also because of the reusability of the components in the verification environment. They are based on the concept of the System Verilog which boosts some of the very essential features like classes, cover-groups, randomization, semaphores, etc. Using UVM and System Verilog, test benches are built which target for coverage of the design.

1.1 FLASH MEMORY: THEORY AND APPLICATIONS

Flash memory is a form of rewritable memory chip which is introduced by Toshiba in 1984.For erase and write mode it allows various memory in one programming operation. Flash memory was developed from EEPROM (electrically erasable programmable read-only memory) and it does not require power supply to hold the contents. It is commonly used in MP3 players, mobile phones, memory cards, digital cameras and USB flash drives.

There are two major types of flash memory called NAND Flash Memory and NOR Flash Memory. The internal characteristics of the NAND and NOR flash memory cells shows similar characteristics of NAND and NOR logic gates respectively.

NAND type flash memory can be read, erased and written in blocks (or pages) which are generally much smaller than the entire device whereas EPROMs has to be erased completely before being rewritten using UV exposure. Blocks of memory in Flash can be easily erased electrically. NOR type flash allows a single byte to be written in an erased location or read independently.

1.1.1 NOR Memories

NOR memories has read mode which is similar to reading from a common memory,which provided address bus and data bus ,so NOR flash memory is much like any address-mapped memory. NOR flash memories can be used as execute-in-place memory, meaning it behaves as a ROM memory mapped.

1.1.2 NAND Memories

NAND memories have finite numbers of write cycles. It has a wear out concept which mean as individual cell performance fails, overall performance degrades. These memories are accessed much like block devices such as hard disks or memory cards. Tunnel injection is used for Writing and tunnel release is used for reading. Read and Programmed mode is done at page level, Erase mode is done at block level.

1.1.3 Difference between NAND and NOR Flash memories

NOR has high read speed and random access capabilities, so it is suitable for code storage. Code storage is present in devices such as PDAs and cell phones.

Conversely, NAND provide fast write and erase capability, it is slower than NOR in the area of read speed. NAND is, however, more than sufficient for a majority of consumer applications such as digital video, music or data storage. Connections are different for both memories. For both memories interface of read and write mode is different. Cost/bit of NAND memory is high, used for file storage.

1.2 INTRODUCTION TO NAND FLASH MEMORY

1.2.1 Principle of operation of a NAND Flash Cell

In flash memory, information is stored in an array of memory cells made from floating-gate transistors. In traditional single-level cell (SLC) devices, only one bit of information is stored in each cell. In multi-level cell (MLC) devices, one bit is stored by each cell.

1.2.2 Floating-gate Transistor

A floating gate transistor (FGT) is a complementary metal-oxide semiconductor (CMOS) technology having two gates: the upper gate is called control gate and the lower gate is called floating gate. Figure 1.1 shows the structure of FG transistor. Both these gates have a thin dielectric material which is used for separation, this layer is called oxide layer. This makes the FG electrically isolated and the electrons placed on it are rapped and remains there. This is the reason behind non-volatile nature of flash memory. The charge in the FG can be modified with the help of extra two small transistors to conduct the tunneling and injection operations. For charge modification applications, the tunneling transistor has to be embedded into a well; hence the technology dictates the type of FGMOS that can be fabricated [6].

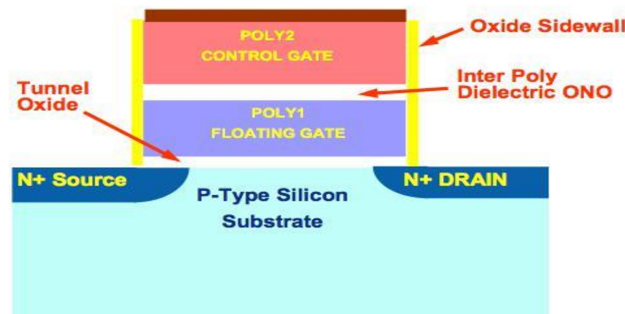


Figure 1 Floating Gate Transistor [1]

1.2.3 NAND Flash Cells are basically of two types

1. ***Single Level Cell (SLC):***

Each cell can store one bit of information hence it can exist in one of two states as given in table 1.1.

Table 1.1: SLC Levels

Value	State
0	Programmed
1	Erased

- SLC memory has faster write speeds and higher cell endurance.
- Lower power consumption.
- This costs more per megabyte of storage to manufacture, since it stores less amount of data per cell compared to MLC
- SLC has longer life and faster transfer speed.
- SLC memory has high operating temperature range.

2. ***Multi Level Cell (MLC):***

- Due to higher data density, MLC has lower cost per unit of storage.
- MLC memory has multiple levels per cell to allow more bits to be stored using the same number of transistors.
- If MLC stores two bits per cell, it can exist in one of four possible states as given in table 1.1.

Table 1.1: MLC Levels

Value	State
00	Fully Programmed
01	Partially Programmed
10	Partially Erased
11	Fully Erased

- Most MLC NAND flash memory can store two bits of information per cell, so it has four possible states per cell.

- Margin separating of the states reduced by multiple states and the possibility of more errors comes. Hence it requires an error correcting code (ECC) that can correct multiple bit errors.

- It has some other drawbacks such as lower number of program-erase cycles, high power consumption compared to SLC flash memory and lower writes speeds.

.

1.3 NAND FLASH OPERATIONS

1.3.1 Program/Write

- The default value in single-level NAND flash cell is binary value ‘1’ value, because under application of an appropriate voltage to the control gate, current will flow through the channel, so that the bit line voltage is pulled down. A NAND flash cell can be programmed to "0" value by following the below step.

- Voltage typically greater than 5 V is applied to the CG and channel gets turned on

- Now the electrons start to flow from the source to the drain (assuming an NMOS transistor).

- Due to hot-electron injection process, electrons gain energy and jump into the FG through the insulating layer.

- Write operation for a NAND Flash is shown in Figure 2.

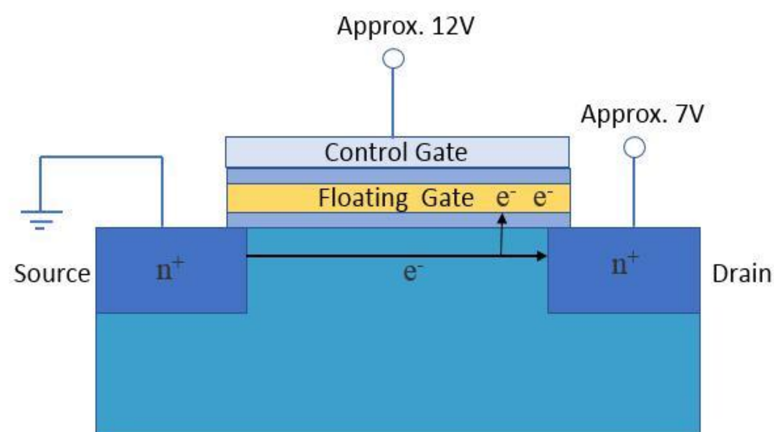


Figure 2 Write operation of NAND flash cell [1]

1.3.2 Read operation

- To read a NAND flash cell, an intermediate voltage (i.e. Verase<V_read_mode<V_program) is applied to the control gate of the cell. Drain is connected to supply voltage and source is connected to the ground. When V_read is applied either one of the below two cases are possible,

- If in erase state, channel is formed and there is a current conduction between drain and the source.

- If in program state, no channel formation and there is very less current (i.e. a leakage current) flows from drain to source. Figure 3 shows the read mode operation of a programmed cell.

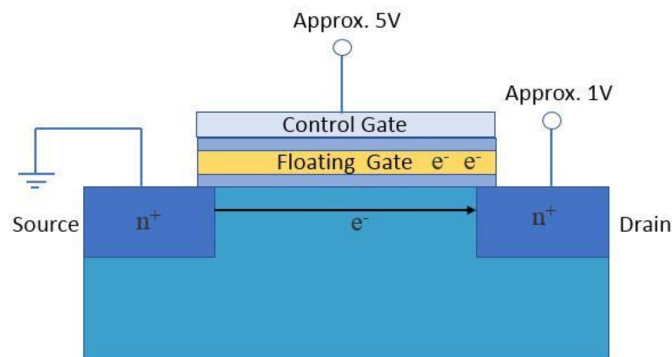


Figure 3 Read mode operation of NAND flash cell [1]

1.3.3 Erase operation

- For erase operation large amount of opposite polarity voltage is applied between the CG and source terminal to pull the electrons off the FG through quantum tunneling. Erasing of NAND Flash cell is shown in Figure4.

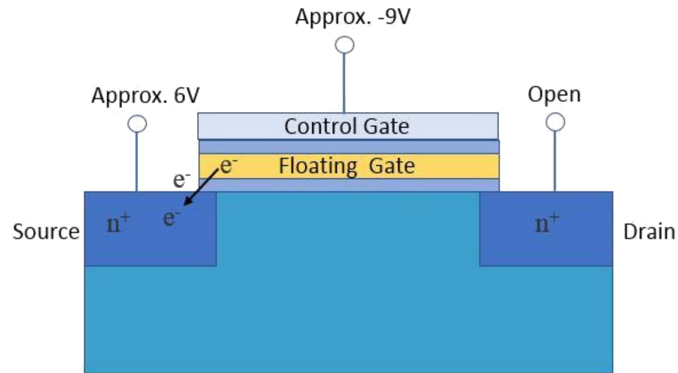


Figure 4 Erase operation of NAND flash cell [1]

1.4 VERIFICATION METHODOLOGIES USED IN INDUSTRY

The purpose of verifying a design is to ensure that it works correctly before it is fabricated and sold to customers. First, the cost of ramping up a design in the fab is very high, so spending the time and money to fabricate a design that turns out to be defective can be financially disastrous. Second, if a defective design gets into customers hands it could adversely affect the reputation of the company and undermine our ability to do business.

In the modern era of large scale Soc, designs are built in pieces (IP blocks) which are assembled in a final integration step. Each piece must be independently verified as well as the fully integrated design. The pieces are often each built by different teams at different times. The essential elements of the digital design methodology include constrained random verification, abstraction, and reusable components. The methodology is implemented in UVM, an industry standard for building test benches using System Verilog. It is currently being developed and maintained by Accellera, an industry standards body. The following is the verification closure criteria for IP:

Functional coverage is 100%. Coverage is developed based on Verification Requirements.

Code coverage is 100% including exclusion list. Exclusion list is set of code coverage points that cannot be covered by simulation. These are reviewed and waived by IP designer.

IP Integration verification is completed in the Core Platform

The project team must define the stability requirement in terms of bug rate that it will enforce. These criteria along with the last four weeks of bug rate data must be presented to the Verification Experts team.

Randomization is done to improve the testing and this is primarily done by creating a random class which will hold the configuration for the current transaction and reusing the same class for the successive randomizations. All configuration register variables are defined as random variables. All sequence control variables are defined in the configuration class and this can be used to create any directed tests, if needed. The sequence of register writing and the delay in between them are randomized to generate a random transaction. Error conditions have a variable defined with unique enum states, and the probability of hitting an error mode will be 1/10. Weightage is assigned to the variables ranges, which are important and those which have higher number of branches.

1.5 TEST BENCH PRINCIPLES

The purpose of a test bench is to determine the correctness of the design under test (DUT). This is accomplished by the following steps:

- Generation of stimulus
- Apply stimulus to the DUT
- Capture the response of DUT
- Check for correctness
- Measure progress against the overall verification goals

Some steps are accomplished automatically by the test bench, while others are manually determined. The methodology chosen determines how the preceding steps are carried out.

The role of a test bench is to stimulate the DUT in different ways and check if the response obtained is correct as expected. Figure mention below gives the test bench design interaction. Modern test benches for SoC designs are written at the transaction level, and the DUT and the test bench interact through interfaces, points of connection through which data and control are transferred. Interfaces form the basis of abstracting the design away from the low-level details of the DUT. Every interaction with the DUT is through interface.

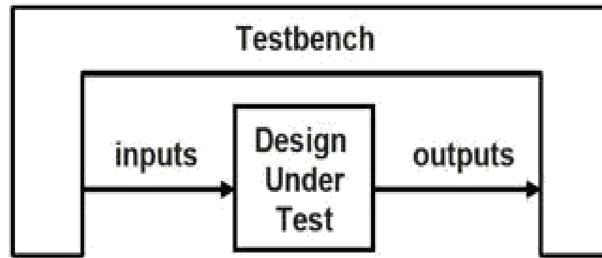


Figure 5Test bench-design environment [2]

1.6 TEST BENCH ARCHITECTURE

Test bench is a group of components performing specific task. The components are shown in Figure 6 .A class is written separately to generate stimulus, drive the test cases, monitor the outputs etc.

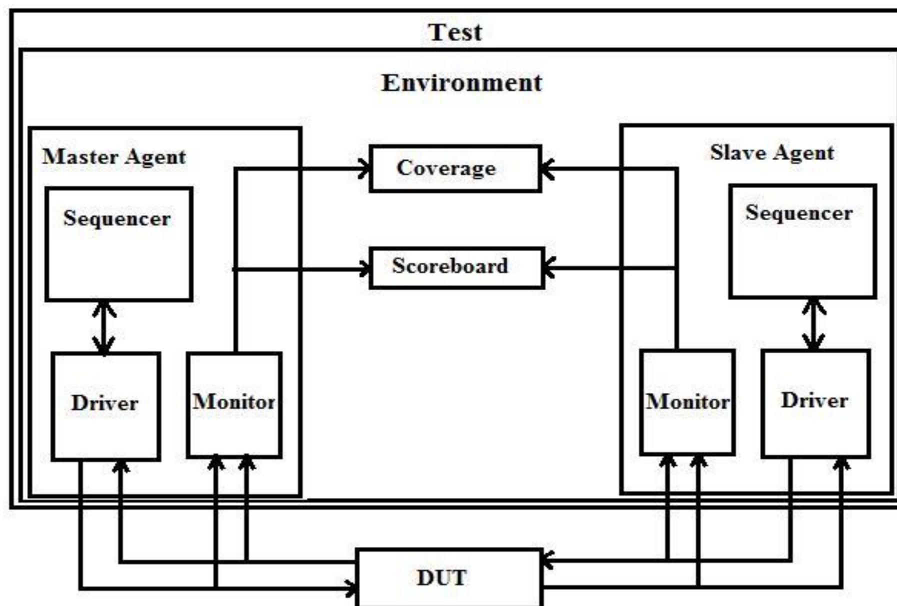


Figure 6Test bench Architecture[2]

DUT is the design written in Verilog or VHDL. All the classes specific to a protocol or interface is grouped by a container class called agent.The components of agent are sequencer, sequence item, driver, monitor and sequence. Sequence will define the sequence in which sequence item i.e. data item needs to be generated and Sequencer will send/receive to/from driver. Driver will get the data packets (sequence item) and translate into suitable form which DUT can understand. The pin level activity on signals in the interface between DUT and test bench.

Environment is observed by monitor and converts it into data packets which are sent to scoreboard. Scoreboard will receive data packets from monitor as well as output from reference model and compares them. Reference model will behave in the same way as the DUT. Agent and scoreboard is grouped within the environment class. Coverage model is a metric which gives the percentage of tested and untested portions of the design. The top most class is the Test. This initiates the stimulus by starting the sequence and also initiates the test bench construction process by building the next level down in the hierarchy.

1.7 COVERAGE CLOSURE

One of the key requirements of verification closure of an IP is to achieve 100% code and functional coverage. Achieving functional coverage (after getting reviewed by designer and architect) is completely under the control of a verification engineer. However, code coverage closure requires couple of iterations with the designer to know which part of code cannot be covered in verification. Examples of these uncovered cases are:

- Code that gets covered only in power aware verification o Code that gets covered only in SCAN mode verification

- Code that never gets hit during verification (aka dead code) o Default case of a fully covered case statement

- Code generated by parameter values between the min configuration and max configuration of the IP

In the above cases, the verification engineer writes an exclusion list and gets a sign off from the designer. The exclusion list is line number based and loses its relevance when the design is updated for comments. One of the ways to handle this is to add pragmas in RTL code for cases where it is 100% clear that it cannot be covered.

1.8 FUNCTIONAL COVERAGE

Functional coverage is one of the important methods which is used to check the functionality of the design. This technique was available around for some time and quite a few companies were actually using this aspect, it has been recently added as a must tool in the process of verification. Functional coverage records some of the metrics like packet length, instruction operation codes, buffer occupancy level etc. to

ensure that the verification of the process is exercised the whole design through all of the required values. But code coverage measures how much of the code is implemented, functional coverage tells about the extent to which the original specifications of the design have been exercised.

1.8.1 Cover points

A coverage point is the sampling of an individual scalar value or expression. The cover points will give the idea of what are the values that are expected to occur in the design and see to it that the verification plan includes those points of interests. The objective of a coverage point is to ensure that all interesting and relevant values have been observed in the sampled value or expression.

A coverage-point bin associates a name and a count with a set of values or a sequence of value transitions. If the bin designates a set of values, the count is incremented every time the coverage point matches one of the values in the set. The bins are written in the cover group as below

```
Covergroup cg;

cover_point_y: coverpoint y {

bins a = {0,1};

bins b = {2,3};

bins c = {4,5};

bins d = {6,7};

}

endgroup
```

1.8.2 Cross coverage

The cross coverage is the method of determining the interdependency between the parameters. This is the important aspect of functional coverage that tells which all signals are interdependent and the time at which the dependency is evident. As seen in coverage points which are concerned with individual scalar values, cross coverage helps in quantification of occurrence of multiple events cross coverage generally involve more than two coverage points. And as a consequence of the crossing the bin number grows

exponentially depending on the number of cover points crossed. The example of cross cover points is shown below.

```
bit [0:1] y;

bit [0:1] z;

covergroup cg;

cover_point_y : coverpoint y ;

cover_point_z : coverpoint z ;

cross_yz : cross cover_point_y,cover_point_z ;

Endgroup
```

Here, in the above program, y can have 4 values 0,1,2 and 3 and similarly z can have 4 values 0,1,2 and 3. The cross product of the y and z will be 16 values (00),(01),(02),(03),(10),(11)........(y,z)......(3,2)(3,3) .

1.9 FORMAL VERIFICATION

1.9.1 INTRODUCTION:

This technique which can be used in various stages of VLSI design flow. Formal verification can be included in front end verification, Logic Synthesis and also back end verification like Post Routing Checks. This is used for checking RTLs design, approach is quite different. Two different types of formal techniques which are available. They are as follows:

1. Formal Equivalence Checking

2. Formal Property Checking

1.9.2 FORMAL EQUIVALENCE CHECKING:

This method is used to find the functional equivalence of a design against the true design. Formal Equivalence checking helps mostly in

1. RTL and Pre-Routed Net list equivalence checking

2. Pre-Routed Net list and Post Routed Net list equivalence checking

3. Net list Vs ECO-Net list equivalence checking

Checking Equivalence is carried out by taking two inputs and results are evaluated by comparing these two signals functionality from the designs i.e., Formal equivalence model and the implemented RTL design. Recently, in industry Combinational equivalence and sequential equivalence checking are used. Equivalence in Combinational circuit is a method where one-to-one mapping of the flip flops between true design and updated design is done. Similarly, In Equivalence checkers of the Sequential circuit verify the structurally different implementations which do not have one-to-one flip flop mapping.

1.9.3 FORMAL PROPERTY CHECKING:

This method has the ability to find the correctness of RTL design or it can even show the root cause of an error by detailed mathematical operations. One of the main advantages of using Formal property checking do the performance without any requirement of test benches or stimuli and time is very less.

This can be implemented by using system verilog assertion languages i.e. ITL Interval Language or Assertion Hardware Descriptive languages. System Verilog Assertion is the subset of the System Verilog language. It is considered to be a significant feature for verification. The primary use of Assertions or properties is to verify the operation of a design is checked by verification tool which in turn can prove whether the design meets its specifications or it violates.

1.9.4 HOW FORMAL VERIFICATION?

Using Formal Verification one can achieve exhaustive verification. Formal verification starts from Initial State, and it explores all possible reachable states in one step until it hits a fixed-point beyond which no new states can be explored. The Formal verification tool tries to validate properties at all stages, thus making it exhaustive.

Assertion/property/feature of SVA has been formally and completely verified and it holds exact result in all possible cases of the design. A failing SVA means that a counter example was found that represents a interruption of the intended design behavior.

1.9.5 DRAWBACK OF SIMULATION FOR FUNCTIONAL VERIFICATION:

Even though, Modern test-bench concepts allow for adaptable and effective modeling and sophisticated coverage analysis, but the simulation by functional verification is still incomplete. This simulation causes high efforts in test-bench design and takes huge amount of simulator run-time.

In test bench creation the stimulus is applied manually which is error prone, time consuming and there are many chances that all cases are not verified. Simulation method of functional verification usually have larger regression runtimes which does not help in meeting the time to market need. Exponential number of test vectors should be exercised to obtain all possible transitions between components.

1.9.6 ADVANTAGE OF FORMAL OVER SIMULATION:

Formal verification method can overcome the insufficiencies of simulation by providing

- Exhaustive verification
- No test bench generation which in turn reduces the Design Verification cycle time
- Exhaustive generation of Stimulus generation is taken care by Formal tool.
- Any illegal input of the design can be provided as constraints. Constraints needed to achieve verification can be included.
- Formal verification provides faster verification. Formal verification tool can complete verification at fraction of time compared to simulation provided no memory blow-up occurs.
- If Properties/Assertions are generic, they can be re-used.
- At the end of formal verification one can achieve high quality RTL and high confidence over the design. Formal verification has better ability to detect errors.

1.9.7 CONS OF FORMAL VERIFICATION

Formal Method of verification have limitation in complexity, time and memory. This issue can be coped up by doing formal verification at modular level rather than whole design block. In Formal verification "Divided we solve, united we blow-up", hence partition should be done to reduce complexity

Other techniques to reduce Complexity includes

- Design Abstraction
- Create abstract model by block removal Environment Simplification
- Tie-offs
- Limiting input space
- Constraints
- Assertion Simplification
- Splitting assertions Parameterization
- Reduce FIFO size
- Concentrate on control logic
- Black Boxing
- Blocks with high complexity but lesser importance with respect to verification goal
- Verifying all transactions of a design at one go can be difficult since it is too large to fit in a single proof window.

1.10 OBJECTIVE OF THE PROJECT

The main objective of this project is to verify the Row Voltage Control Block in NAND Flash Memory. Functional coverage model is developed for checking the functional correctness of the design. The design is verified in the verification environment with this model and the functional coverage percentage is found. If the functionalities are not covered completely, then test cases from uncovered report are generated and regression is done to achieve 100% percentage coverage.

1.11 THESIS ORGANISATION OF REPORT

The thesis is organized consisting of 6 chapters. Chapter 2 deals with the literature review on the existing methodologies for verifying the RTL design. Chapter 3 explains the working of module inside NAND Flash Memory called Row Voltage Control Block and steps to verify the module. Implementation of the proposed verification methodology and results of the verification is discussed in the chapter 4. The report ends with the concluding remarks in the last chapter.

CHAPTER 2

LITERATURE SURVEY

The complexity of today's multi-core system on chips (SoCs) can be addressed if the main challenges in verification are handled successfully. Verification of digital designs play a very important role in present day chips because of the increase in complexity of the design. New methodology for verification like UVM is gaining popularity because of the support extended by the major EDA tool vendors and also because of the reusability of the components in the verification environment. With increasing adoption of UVM, each intellectual property (IP) could be well verified in a block level UVM test bench.

Eman El Mandouh et al., in [1] uses formal methods to automate the identification of uncovered cover points and then automatically generates exclusion of these cover items to get final accurate results. In order to generate these test scenarios formal verification is used so that the goal is achieved by reaching the cover points which didn't get covered in the initial run.

Alfonso Martinez Cruz et al., [2] has talked about a verification method which uses Binary Differential Evolution algorithm to get lists of all the vectors that can maximize the functional coverage in Synchronous First-In-First-out (FIFO) memory. The results obtained in this has been proven to effective high functional coverage percentage i.e. more than ninety percent as compared in different cases.

Harry Foster in [3] describes the application of assertion based verification which addresses the 2 main problems in verification, namely, poor controllability and poor observability. It provides a step-by-step approach to create an assertion-based IP. An in detail example of bus protocol is considered, which brings out different concepts illustrating an effective methodology to develop assertions and assertion-based verification IP.

Eman El Mandouh et al., in [4] has provided a solution of automatic extraction of assertions from simulation traces. It combines the search for known assertions via frequent template and sequential pattern mining. Static analysis methods are used where constraints are automatically extracted from Register Transfer Level (RTL) design. This approach has helped in assertion detection which are typically common and widely used in RTL design.

Khalifa introduced an architecture which is built based on UVM for verifying Flash memory controller designs [5]. The base architecture is used to create a configurable abstraction verification environment for Flash memory controller designs. It can also help in integration of the essential UVM Verification Components (VCs). The paper guides the verifier where and how to put the behavior of Flash memory controller in its correct and suitable place in the architecture so that it can also serve the idea of reusability.

S. Asaf et al., describes the coverage analysis to improve the quality of verification process. Reports from coverage tools tells us about the areas in the design that are not tested thoroughly[6]. Also due to increased sizes ,the analysis of large coverage models takes more time. He introduced a method on cross product functional coverage.This method allows user to improve the quality of the coverage to higher extent.

Jusas et al., in [7] proposed a semi-automatic framework which generates stimuli for tests used in parallel VHDL designs. The framework was experimented on 2 designs – GCD implementation in VHDL and B13 circuit. The generated test vectors are synthesized for a particular FPGA family and verified on FPGA boards. Behavioural simulation is carried out and the code coverage obtained is verified using Xilinx FPGA hardware. The code coverage metrics used are statement, branch and toggle coverage. The statement and line coverage is 100% for both the designs whereas toggle coverage for GCD and B13 design is 100% and 27% respectively.

Laszlo Molnar et al., in [8] gives a brief explanation of the evolution of functional simulation methods beginning from late 90's until recent due to the increase in the complexity of the design. Simulation of ASICs is classified into 3 categories, namely, Fault, Functional and Timing. The initial models used in the late 90's were based on VHDL and Python. Due to the growth in complexity of digital designs, new verification methodologies like Open Verification Methodology (OVM) and successively, UVM came into existence. These are based on System Verilog which supports some of the very essential features like classes, cover-groups, randomization, semaphores, etc. Using UVM and System Verilog, test benches which target for coverage can be built.

Challenges in SoC Verification and Best Practices are presented by Prokash Ghosh et al., in [9]. Different verification strategies like Pin multiplexing flow, low power flow, pipeline flow, register space flow, etc., are implemented on the existing conventional methodologies. The results obtained are remarkable in terms of the cycle time. The final production qualification cycle was pulled in almost by 3 months. The methodologies are found to be a great success. Challenging areas are considered and major bugs are reported using the proposed methodologies. A brief overview of the best practices to be followed is also included.

Wei Ni et al., in [10] worked on Functional Coverage-Driven UVM-based UART IP Verification. UVM-based technology can significantly reduce the time which is to be required. A UVM-based platform featuring functional model which based on coverage is built to find out whether the verification which achieves the expected output. This is found to achieve 100% functional coverage with the coverage method by adding test vectors and using constrained random test. Simulation results showed UVM methodology could be used to verify the UART IP and SoC featuring UART interface.

Increase in complexities of hardware designs has made exhaustive simulation of designs very difficult. This has increased the need for an additional complementary verification technique. This need has been satisfied by the use of formal analysis on industrial hardware designs. Formal analysis of hardware design uses mathematical techniques to prove whether the design implementation confirms to the specification. The specification can be stated as a set of features which should hold on the design under verification. In this reference [11], various methods applied in formal hardware verification, and various automated techniques to make the verification error free and even techniques to handle the state explosion problem have been stated. Application of formal analysis techniques on appropriate designs is key to successful verification.

2.1 STATEMENT OF PROBLEM BASED ON IDENTIFIED RESEARCH GAPS

Simulation, is one of the oldest verification method used. This method is extensively used in the industries to verify their chip designs, the functionality of the chip. Although it has many advantages but when used on large scale applications it suffers through some failures including consumption of large time and as we know time is equal to money when seen from an industry perspective, so we cannot compromise on time and which means minimum time should be used during simulations.

Another issue is related to the errors found while doing simulation. The errors/bugs found in the design are difficult to identify. One more issue we need to consider here is the reusability of the design.

Reusability of the design not only decreases our effort but also increases the efficiency by taking less time. The efficiency of simulation process can be improved from three aspects which as follows:

Firstly, we require improving the verification based test vectors' pertinence. The traditional coverage based on code verification can't show the effective relationship between coverage and functional points, so the vectors are poor at pertinence. With the increasing scales, the defects become more and more evident.

Secondly, we need to run simulation automatically in order to verify a large scale design, we need more test cases.

Write all the test cases by hand is not only slow but easy to miss some corner cases. To avoid missing of test cases we use automation by using scripting.
Furthermore, to check the simulation results is very lengthy process. So, we have to do the verification by generating the various test vectors, checking the results and configuring the verification environment .

Thirdly, we will discuss about reusability. For the large scale design we are using the UVM methodology where we use reusable test benches which takes less effort as compared to non-reusable environment and also decreases the workload and increases the efficiency. If little parts of the verification environment is reused between different modules or between module and system or in between the projects , then workload is reduced.

CHAPTER 3

VERIFICATION OF MODULES INSIDE NAND FLASH MEMORY

While accessing the particular word line in memory, the neighboring word lines should have particular voltage levels, so that the intended operation completes successfully without any error. A block in memory called Row Voltage Control Block handles this by supplying suitable voltage signals to the word lines based on some conditions.

3.1 ROW VOLTAGE CONTROL BLOCK

The main function of Row Voltage Control Block is to generate voltage control signals for each data latch operation .Figure 3.1 represent the block diagram of Row Voltage Control Block. In the row block of the NAND flash memory, we have the different parameter signal, command signal on the specific operation which has to be performed. When all these conditions are applied to the NAND flash memory, this will produce the voltage control signal at the word line of the memory.

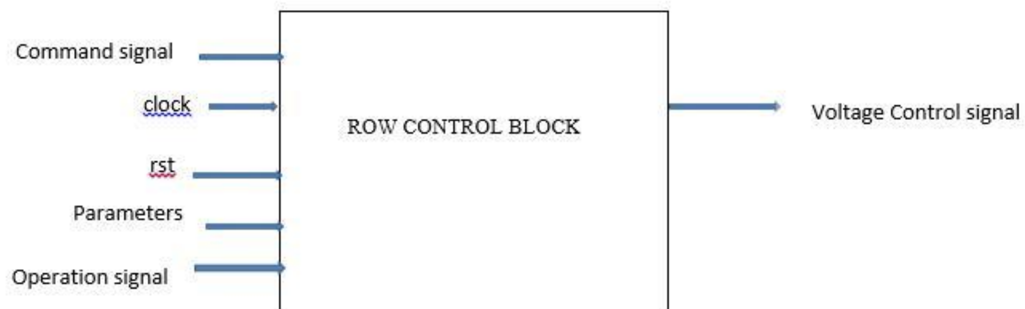


Figure 7 Row Voltage Control Block

For Row Voltage Control block we have the executable specification format, which is the input file where all the signals are given with the particular value in Figure.

D10 | voltage signal9

	A	B	C	D	E
1		operation signal	operation signal	operation signal	
2					
3		parameter1	parameter 4	parameter7	
4		parameter2	parameter 5	parameter 8	
5		parameter 3	parameter 6	parameter 9	
6					
7					
8	command signal 1	voltage signal1	voltage signal 4	voltage signal 7	
9	command signal2	voltage signal2	voltage signal 5	voltage signal 8	
10	command signal 3	voltage signal3	voltage signal 6	voltage signal9	
11					
12					
13					
14					

Figure 8 Executable Specification Format

In NAND Flash Memory, we have operation modes like Read, Write, and Erase etc. For all the operation signals we have different parameter values. If these parameter values are enabled with the particular command signal, then corresponding voltage signal is applied to word lines of NAND flash memory.

3.2 VERIFICATION FLOW CHART

Below figure represents the verification layout flow. From Design specifications, test bench is developed and it is been integrated with RTL for checking. Failures are checked and if any, RTL is modified. If there exists no failures, code coverage is analysed first. Test bench is modified if 100% coverage is not achieved. If all the lines in the test bench are covered, then functional coverage percentage is analysed. If the functionality is not fully covered, again test bench is modified by adding more test cases until 100% is attained.

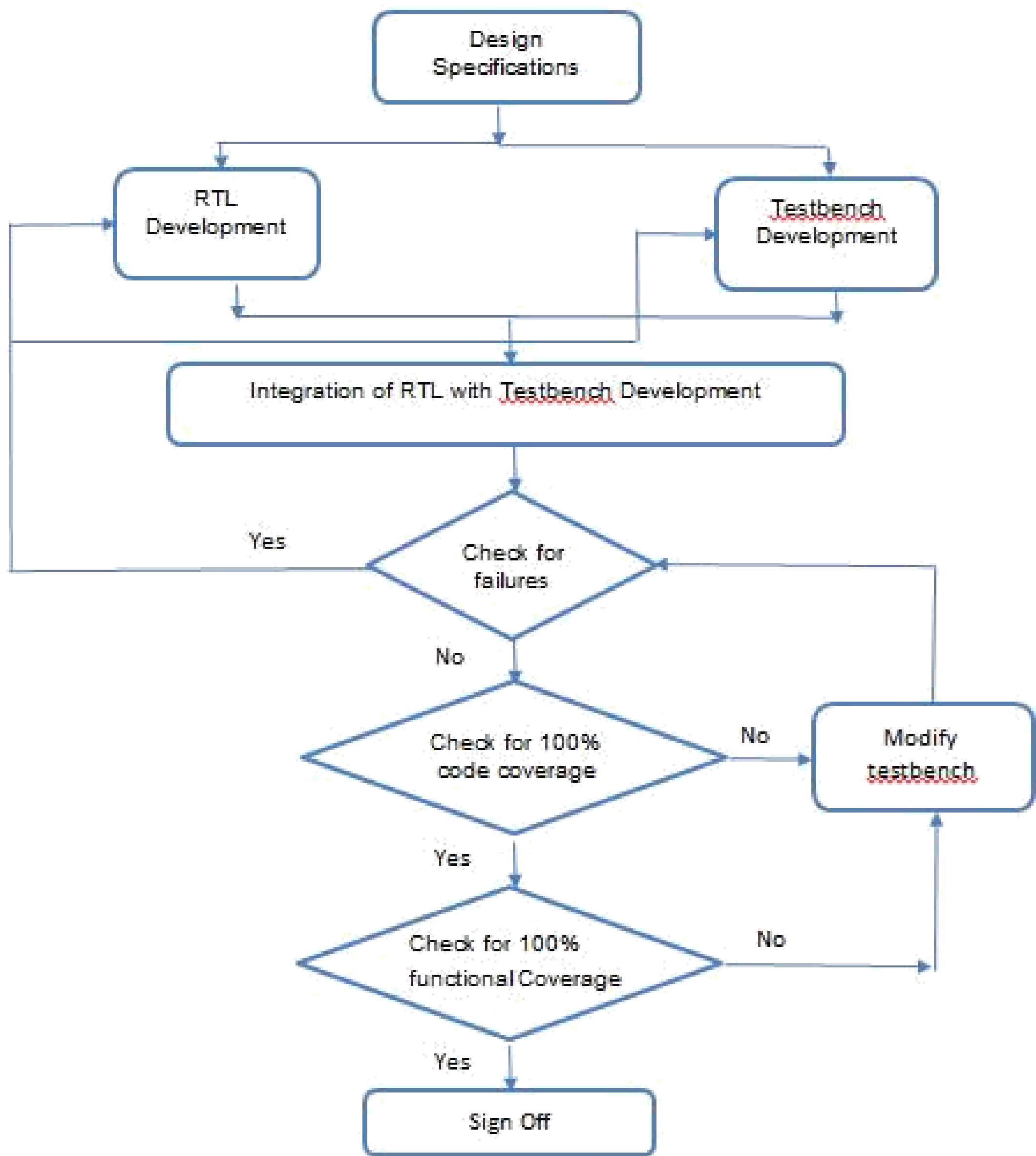


Figure 9 Verification Flow Chart

3.3 VERIFICATION STEPS

Below are the steps to be followed for verifying Row Voltage Control Block

1. Verilog and UVM are used to build a consistent verification environment (provide test benches to run the developed assertions).

2. Scripts are used to provide automated and generic verification flow of the design.

3. Assertions are used to fast the test based on various modes and functions and hence increasing the quality of the design.

4. Constrained-random test generation is used to achieve high coverage and the rest corner cases verified by directed tests.

5. Regression of the test vectors and bug issues are used to repeat problem and progress of fixes the issues.

CHAPTER 4

IMPLEMENTATION OF VERIFICATION

METHODOLOGY AND RESULTS

To check for the extent to which design specifications have been exercised, functional coverage is written where the user can specify the functions or modes to be analysed as cover points. From the design specifications, we can know which are the different operation codes and corresponding parameters to be checked. So we will be writing cover points for each operation codes and storing the number of times they are hit in "bins". Also user can see whether all combinations of parameters corresponding to each operation codes is hit or not by specifying cross cover of parameters.

4.1 COVERAGE MODEL

The coverage model is represented in figure 4.1. These steps are involved in Functional Coverage methodology are:

1. Creation of Functional coverage model from specification
2. Separation of unique cover points from coverage model
3. Test bench is simulated with coverage model and unique cover points. The tool generates report with covered and uncovered bins.
4. From the report generated by tool, uncovered report is extracted with the help of script.
5. Generation of test vectors from uncovered report and running regressions with test cases.
6. Analyzing coverage reports and including new test cases to get better coverage values.
7. Once the coverage reaches 100% or close to 100%, the design is cleared for tape out.

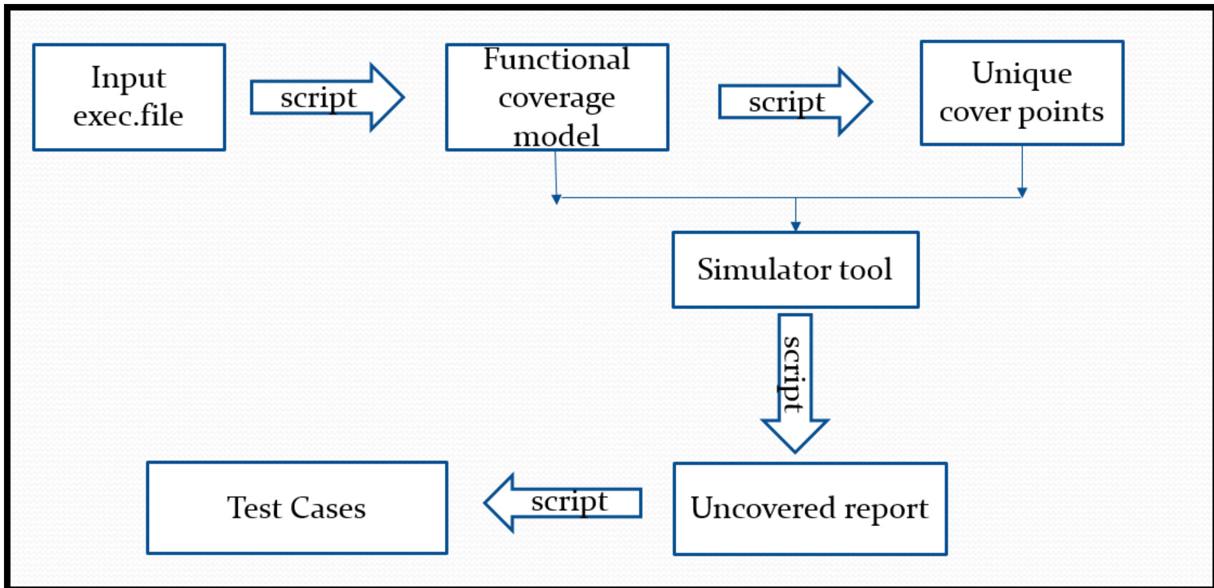


Figure 10 Coverage Automation Flow

For Row Voltage Control block we have the executable specification format, which is the input file where all the signals are given with the particular value and conditions in Figure 4.

	A	B	C	D
1				
2	operation signal		ERASE	ERASE
3	Parameters/CMD signals		cmd 1	cmd 1
4			F_param1	F_param1
5	Voltage signals	WL_X	V1	V1
6		WL_y	V2	V2
7		WL_z	V3	V3
8				
9				
10	operation signal	Erase/Read/Program		
11	Parameters/CMD signals(input)	F_param1		
12		cmd1		
13	Voltage signals(output)	WL_x		
14		WL_y		
15		WL_Z		

Figure 11 Input Executable File

Operation signal in NAND Flash Memory are Erase, Read and Write. For these parameter signals, there will be different parameter signals. Parameter signals and command signals are f_param1 and cmd1.From the input specification format if the parameter signals and command signals with particular value are given to the NAND Flash Memory it would generate the voltage signal on the word lines which are define here by WL_x,WL_y,WL_z.

4.1.2 Functional coverage model

This model is observe the functionality of a test vectors. As such, it is code is written in the automatic manner which is used to track the important values, sets of values, which corresponding to design, features, or boundary conditions have been exercised. This is very important in the verification approach this is the factors which used to determine whether testing is done.100% functional coverage indicates that all the functionality of the test plan have been exercised properly. Fig 4.3 shows the functional coverage model. Here we make the cover group which contains the cover points. Each cover point is associated with the particular bin values. Then crosses between those cover points are made.

```
Class cover_model ;
Covergroup ERASE @(posedge clk);
Cov_point1 : coverpoint cmd_1( )
  {          bins_c0={0};
              bins_c1={1};
   }
Cov_point2: coverpoint     param_1[1:0]

  {          bins_f0={0,1};
              bins_f1={2,3};
   }
cov_point3: coverpoint v1[2:0]

    {        bins_bin0=[0:2];
             bins_bins1=[3:5];
             bins_bins2=[6:7];
     }

cov_point4: coverpoint v2[1:0]

    {        bins_bin0=[0:1];
             bins_bins1=[2:3];

     }


cov_point3: coverpoint v

    {        bins_bin0={0};
             bins_bins1={1};

     }

cross c1: cross cmd_1,   param_1;
cross c2: cross v1,v2;
end cover_model
~
~
```

Figure 12 Functional coverage model

4.1.3 *Unique cover points:*

In functional coverage model, there will be multiple copies of cover points. To make it unique, we write a script to separate the unique cover points and crosses as two different files. In unique cover point file which is shown in fig 4.4, we have the signals which are assigned to the particular bins.

```
cmd 1:                coverpoint sig.cmd 1 {bins ={0:1};}
 param1:              coverpoint sig.  param1{bins one={0:2};}
V1:                   coverpoint sig.V1{bins={0:2};}
V2:                   coverpoint sig.V2{bins={0:2};}
V3:                   coverpoint sig.V3{bins={0:1};}
V5:                   coverpoint sig.V5{bins={0:1};}
V6:                   coverpoint sig.V6{bins={1};}
V7:                   coverpoint sig.V7{bins={0:2};}
V8:                   coverpoint sig.V8{bins={0};}
V9:                   coverpoint sig.V9{bins={1};}
V10:                  coverpoint sig.V10{bins={1};}
cmd 2:                coverpoint sig.cmd 2{bins={1};}
```

Figure 13 Unique cover point

4.1.4 *Uncovered Report*

Verification is done in a constrained-random verification environment. The challenging part in simulation based functional verification is to generate tests to achieve high coverage. Based on the constraints, test cases are generated and supplied to the design. These cases may not hit all the bins created in the coverage model which means it may miss to test all the functionalities of the design under verification.

```
|--v1                          0.00% (0/1)        v1:                coverpoint sig.v1 {bins one = {1};}
| |--one                       0.00% (0/1)        v1:                coverpoint sig.v1 {bins one = {1};}
|--cross_param1               50.00% (1/2)        cross_param1 :      cross param2, param3;
| |--one[1],one[1]             0.00% (0/1)        cross_param1 :      cross param2, param3;
--MODE_1                       0.00% (0/1)        MODE_1:cross param2,v1,v2,v3;
| |--one[0],one,one,one 0.00% (0/1)               MODE_1:cross param2,v1,v2,v3;
|--MODE_1_0                    0.00% (0/2)        MODE_1_0: cross MODE_1, param3;
| |--(one[0],one,one,one),one[0] 0.00% (0/1)      MODE_1_0: cross MODE_1, param3;
```

Figure 14 Uncovered Report

The test bench is simulated with the functional coverage model and the unique cover points in the simulator tool. From the report generated by simulator tool, script is written to form uncovered report. Figure 4.5 shows the format of uncovered report.

4.1.5 Test cases

Test cases are generated based on the bin values in the uncovered report using the script. Using these test cases, regression is done and analyzed the functional coverage percentage. The format of test cases used for running regression is shown in figure.

```
{
v1=1'h1
}
{
param2=1'h1
param3=1'h1
}
{
param2=1'h0
v1=1'h1
v2=1'h1
v3=1'h1
}
{
param2=1'h0
v1=1'h1
v2=1'h1
v3=1'h1
param3=1'h0
}
```

Figure 15 Test cases

4.2 RESULTS

Above figures shows the coverage results obtained for erase, program and read modes before running regression. The coverage results obtained after running regression with test cases taken from uncovered report is shown in figures.

Figure 16(i) Coverage results for erase before regression

Figure 17(ii) Coverage results for erase after regression

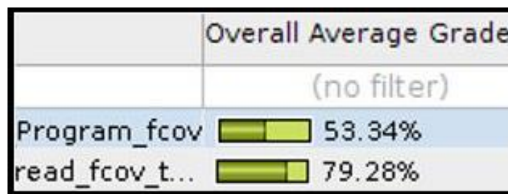


Figure 18(i) Coverage results for Program and Read before regression

4.2.1 Comparison of coverage percentage

Table 4.1 Comparison of Coverage Percentage

Mode	Before Regression	After Regression
Erase	61.03%	98.92%
Program	53.34%	98.64%
Read	79.28%	98.83%

It is evident from the results in the table that coverage percentage is increased after running regression with test cases generated from uncovered report.

CHAPTER 5

FORMAL VERIFICATION OF OPERATION MODULE

5.1 INTRODUCTION

NAND Flash memory has many modules which perform the intended functions. Before the design is taped out for manufacture, it is important that the design should be functionally verified. In case of Verification, Verification engineers should perform thorough verification to ensure that the RTL performs the intended functionality. Manual intervention of writing test benches to provide inputs are mostly randomized and hence it can capture the interesting cases but the verification cannot be done exhaustively. In order to do an exhaustive verification Formal verification tools provide all needed inputs which helps in verification. At the end of Formal verification our intent is to achieve a 100% clean RTL design which performs the intended functionality.

5.2 NEED FOR COMMAND MODULE

Flash memory is under control by using multiple of commands. The sets of commands are different based on mode to mode performance. There are many commands, few of them are universally to many NAND Flash manufacturers which are manufacture specific and supported by a less devices. A standard NAND Flash Interface has a some of basic mandatory command set. The most common commands are ´program´, ´read verify data´, ´erase mode wise´, ´reset based´, ´program confirmation´, ´read_pvfy´ . Device specific commands contain ´random read operation´, ´page cache read operation´, ´random write´, ´page cache write´, ´internal data move´, ´two-planes write´, ´two planes read´, and so on. NAND flash devices carry out three basic operations: program based on page level, erase based on block level, and read based on page level. In this device read mode and program mode performance take based on a page basis rather than on a byte or word basis like NOR Flash.

NAND flash memory has several mode and under each mode several flavours of operation are performed. The main flavours of operation include

- Read
- Erase
- Program

Eight bit operation codes are used to identify a mode and their respective operations. If a particular Operation code is exercised the corresponding signal gets asserted. The codes along with favourable conditions can either assert an operation signal or De-assert an operation signal.

The Operation module RTL architecture includes 3 types. And each type of architecture requires unique assertion to check the functionality. The overall verification flow is picturized in the form of a flow diagram in Figure 19.

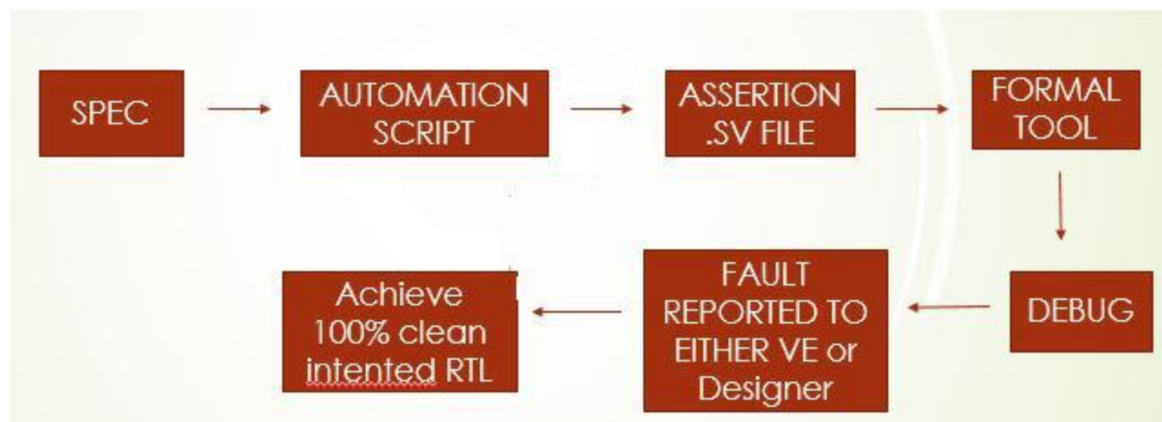


Figure 19 Verification flow diagram

5.3 ARCHITECTURE AND ASSERTIONS

The RTL module implementations are mostly done using registers. Based on the operations performed in registers they are divided into 3 Flavors. The conditions which are favorable for a signal to be asserted

includes Enable, Data and Conf whereas, the favorable condition to de-assert a signal are based on Sync Rst and Async rst.

5.3.1 FLAVOR1 & FLAVOR2

The architecture of Flavor 1 and Flavor 3 is same as the architecture of normal register. The only difference between Flavor 1 and Flavor 3 is the availability of conf signal. Same assertions are used for Flavor 1 and Flavor 3. For Flavor 3 commands, the conf signal is tied to 1'b1. If there are favourable configurations such as the expected code along with conf, enable and without Sync rst the signal gets asserted.

5.3.1.1 ASSERTIONS FOR CHECKING SIGNAL ASSERTION

```
property Flavor_1_3(clk,async_rst,code,enable,conf,sync_rst,signal); @(posedge clk) disable iff (!async_rst)

(code  && enable  && conf && !sync_rst) |=> signal;

Endproperty
```

```
property Flavor_neg_1_3(clk,async_rst,code,enable,conf,sync_rst,signal); @(posedge clk) disable iff (!async_rst)

$rose(signal)  |-> ($past(async_rst) == 'b1) && $past(enable) && $past(conf)&& $past(code)

$past(!sync_rst); Endproperty
```

5.3.1.2 ASSERTIONS FOR CHECKING SIGNAL DE-ASSERTION

```
property Flavor_deassert1_3(clk,async_rst,data,enable,conf,sync_rst,signal); @(posedge clk) disable iff (!async_rst)
```

(sync_rst) |=> !signal;

property Flavor_neg_1_3(clk,async_rst,code,enable,conf,sync_rst,signal); @(posedge clk) disable iff (!async_rst)

$rose(signal) |-> ($past(async_rst) == 'b1) && $past(enable) && $past(conf)&& $past(code)

$past(!sync_rst); Endproperty

5.3.2.1 ASSERTIONS FOR CHECKING SIGNAL ASSERTION

property Flavor_2(clk,async_rst,code,enable,signal); @(posedge clk) disable iff (!async_rst)

(code && enable) |=> signal;

endproperty

property Flavor_2_neg(clk,async_rst,code,enable,signal); @(posedge clk) disable iff (!async_rst)

$rose(signal) |-> $past(code & enable);

Endproperty

5.3.2.2 ASSERTIONS FOR CHECKING SIGNAL DE-ASSERTION

property Flavor_2_deassert(clk,async_rst,code,enable,sync_rst,signal); @(posedge clk) disable iff (!async_rst)

(!code) |=> !(signal);

endproperty

5.3.3 ASSERTIONS TO CHECK ASYNC RESET

property chk_async_rst(clk, async_rst,signal); @(posedge clk)

(!async_rst) |-> !signal;

Endproperty

5.4 SPECIFICATION DOCUMENT

The verification should be performed by the verification engineers using the data available to them from the SPEC document. Because if the verification is done from designers interpretation then there are more chances that the intent of the design which is expected is not verified.

For Operation module the SPEC document has the code along with asserting and de-asserting signals. The format of the SPEC document is shown in Figure 20.

Operation CODE	SIGNAL	ARCHITECTURE	ENABLE	CONF	SYNC_RST	ASYNC_RST
		1				
		2				
		3				

Figure 20Specification Format

5.5 NEED FOR AUTOMATION

Each Operation code has nearly 6 assertions. And writing assertions for each command manually involves a lot of human effort and there any more chances that it is error prone. In order to reduce the effort, automation script is written which extracts the respective signals from the SPEC document and it can be used in the assertions. The automation script is generalized which can be used for writing assertions for any variant of Operation module.

At the end of running the automation script with SPEC document as input we get an .sv (assertion file) which is later provided as input to the Formal verification tool. The contents of the assertion file are shown below.

```
module check( input ports);
wire      ;
wire      ;  //Wire declarations
Properties          //Properties written by verification engineer in the verification plan


:
assert property(); //assert statements to assert the properties

endmodule

bind // Helps in binding the inputs of module to the Design file which helps the Formal verification tool in driving the inputs.
```

5.6 FORMAL VERIFICATION

Formal verification tool is loaded with .tcl file which provides inputs like Design files, Library files which are compiled, and then elaborated. In order to set the initial condition, the needed clocks are provided with a clock period and a reset sequence is provided. Once the initial conditions are set, the .sv files which have assertions listed is provided as input.

Sometimes we must constraint the tool from providing certain inputs which are more like illegal inputs. Hence an assumption file is also provided as input in .tcl file. The .tcl file with these inputs is loaded into

the Formal verification tool. Once the input file is loaded into the formal verification tool, the needed inputs are provided by the tool and the tool tries to find a case where the assertion fails. If the assertion fails the "Debug" option is enabled by the tool. With the debugger option available the reason for the failure of assertion can be found. The reason for the failure of assertion could be either the understanding of design and Spec is misaligned and the assertion is written or the RTL is behaving in a faulty way. If the feature is misunderstood by the verification engineer then assertions should be changed accordingly else if the RTL is not behaving in intended way the issue should be reported to the designer. At the end of Formal verification, a 100% clean intended RTL is obtained.

5.7 DEBUG RESULTS

sva/ assertion1	hold
sva assertion2	hold
sva/ assertion3	fail (1)
sva/ assertion4	fail (2)
sva/ assertion5	hold
sva/ assertion6	hold
sva/ assertion7	hold
sva/ assertion8	hold

Figure 21Results before Debug

Assertions	
sva/assertion1	hold
sva/assertion2	hold
sva/assertion3	hold
sva/ assertion4	hold
sva/ assertion5	hold
sva/ assertion6	hold
sva/ assertion7	hold
sva/ assertion8	hold

Figure 22Results after Debug

The total number of assertions listed includes 1535 and all 1535 listed got passed thus preserving the intended features of the RTL. Thus, achieving a 100% clean intended RTL. Verification is done completely for the module.

5.8 REASONS FOR ASSERTION FAILURE

The reasons for the failing assertion are listed below

- Mismatch of SPEC and RTL
- Without the required command or favourable condition, the signal getting asserted
- Any unintended behaviour of RTL
- Unimplemented signals in RTL but provided in SPEC.
- If a Signal doesn't arrive at all in RTL then vacuous pass will be attained.

5.9 CONCLUSION:

The reasons for all the failing and the vacuity was debugged and analysed. The issues were reported to the designer and the corresponding solutions were either provided by the designer or based on the RTL functionality suitable constraints (assumptions) were written by the verification engineer to achieve 100% clean intended RTL.

CHAPTER 6

CONCLUSION

Verification is one the major problem in the design of silicon-chip devices and the reusability of the IP blocks. As design complexity is increased, the new techniques are now emerging which are proved to be effective for appropriate contribution and deployment. The Silicon chip industry require a reuse concept, verification methodology based on coverage which built System verilog standard language.

Verification of modules in NAND flash memory called Row Voltage Control Block is done in this project. Verification derived by coverage using Universal Verification Methodology is used in the verification of the Row Voltage Control Block. This essentially required the writing of test bench, creation of the score board, writing of the cover groups and writing of the test cases to get all the bins covered. The functional coverage is more than 95% which is essential condition for tape-out of the design. The improved coverage percentage is obtained by running the regressions with different test cases driving the DUT. The test cases are modified after careful analysis of the report generated by the simulation tool. Thus verification of Row Voltage Control Block is successfully signed off for tape-out.

BIBLIOGRAPHY

[1] http://www.hitequest.com/Kiss/Flash_terms.htm

[2] Chris Spear and Greg Tumbush, ”SystemVerilog for Verification: A Guide to Learning the Testbench Language Features”, pp. 1- 436, Springer Publishing Company, 2009.

[3] El Mandouh and AG Wassal, “Automatic generation of functional coverage models”, IEEE International Symposium on Circuits and systems, pp. 754-757, 2016.

[4] Alfonso Martinez Cruz, Ricardo Barron, Fernandez and Heron Molina Lozano, “Automated Functional Coverage for a Digital System Based on a Binary Differential Evolution Algorithm”, IEEE Computer Society, pp. 92-97,2013.

[5] Harry Foster, "Applied Assertion-Based Verification: An Industry Perspective", Foundations and Trends in Electronic Design Automation, vol. 3, no. 1, pp. 1-95, 2009.

[6] El Mandouh and AG Wassal, “CovGen: a framework for automatic extraction of functional coverage models”, International Symposium on Quality Electronic Design, pp. 146-151, 2016.

[7] Khaled Khalifa, “Extendable generic base verification architecture for flash memory controllers based on UVM”, IEEE International Conference on Computer Supported Cooperative Work in Design, pp. 584-589, 2017.

[8] SigalAsaf, Eitan Marcus and AviZiv, ”Defining coverage views to improve functional coverage analysis”, Proceedings of the 41st Annual Conference on Design Automation, pp. 41-44, 2004.

[9] VaciusJusas and Tomas Neverdauskas, ”Stimuli generation framework for testing multiple processes in VHDL”, Information Technology and Control, pp. 440-446, 2014.

[10] L. Molnar and A. Gontean, "Functional Simulation methodes," *IEEE International Symposium on Electronics and Telecommunications (ISETC),* pp. 198-202,2016.

[11] P Ghosh, S Ghosh, P Singh and S Mishra, ”Case study: Re-visiting SoC verification challenges and best practices”, IEEE International Symposium on VLSI Design, pp. 1-9, 2015.

[12] Wei Ni., and Xiaotian Wang, ”Functional coverage-driven UVM-based UART IP verification”, IEEE International Conference on ASIC, 2015.

[13] Praveen Tiwari, Raj Mitra, Manu Chopra and Alok Jain, "Tutorial T4B: Formal AssertionBased Verification in Industrial Setting", IEEE 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems, Bangalore, India, June. 2007.

.

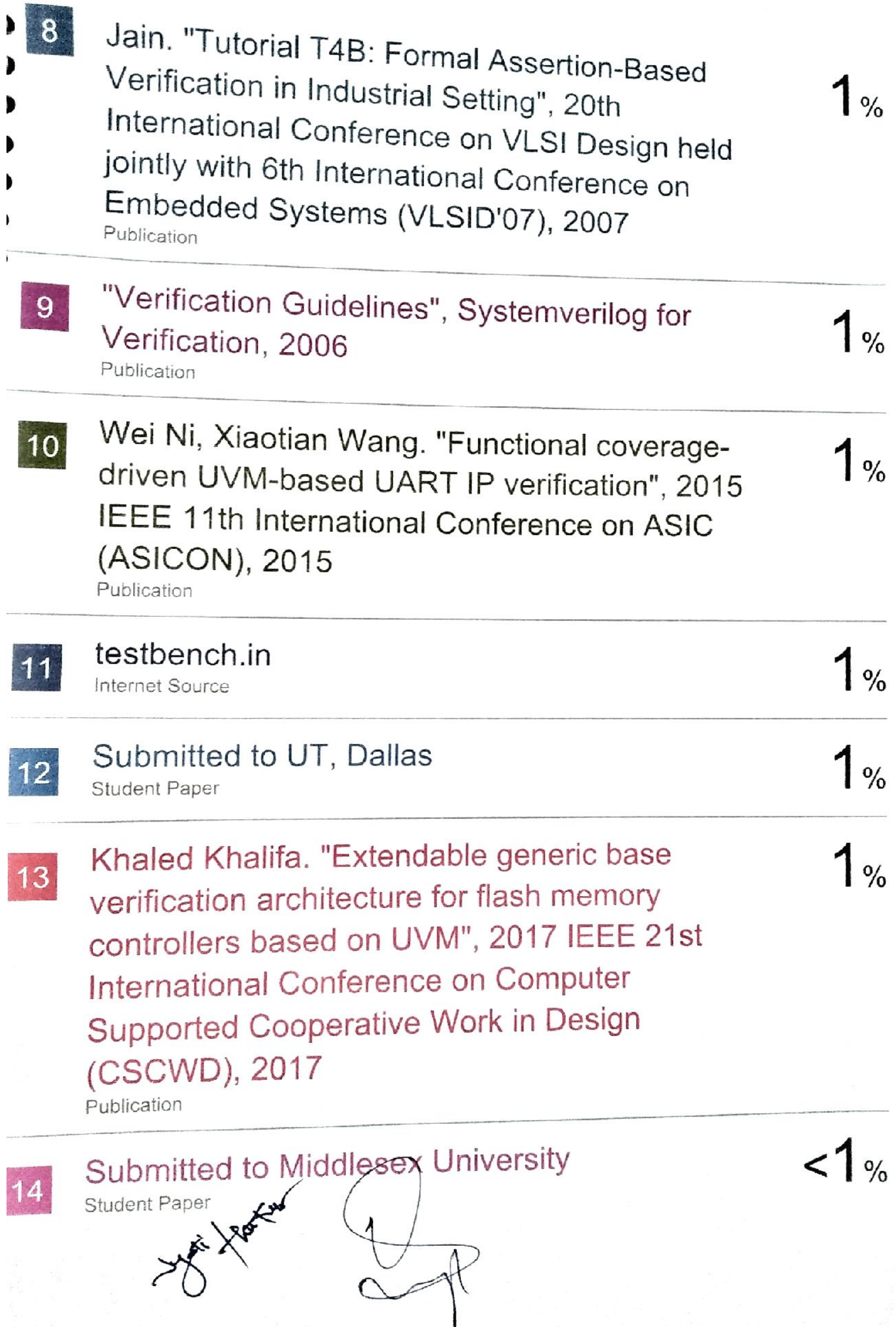
ORIGINALITY REPORT

17%	12%	8%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

#	Source	%
1	Runshan Yang, Liji Wu, Jun Guo, Baorong Liu. "The research and implement of an advanced function coverage based verification environment", 2007 7th International Conference on ASIC, 2007 Publication	1%
2	www.tutorialsweb.com Internet Source	1%
3	Submitted to IIT Delhi Student Paper	1%
4	vlsi.pro Internet Source	1%
5	docslide.us Internet Source	1%
6	www.omicsonline.org Internet Source	1%
7	en.wikipedia.org Internet Source	1%

Praveen Tiwari, Raj Mitra, Manu Chopra, Alok



8	Jain. "Tutorial T4B: Formal Assertion-Based Verification in Industrial Setting", 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07), 2007 Publication	1%
9	"Verification Guidelines", Systemverilog for Verification, 2006 Publication	1%
10	Wei Ni, Xiaotian Wang. "Functional coverage-driven UVM-based UART IP verification", 2015 IEEE 11th International Conference on ASIC (ASICON), 2015 Publication	1%
11	testbench.in Internet Source	1%
12	Submitted to UT, Dallas Student Paper	1%
13	Khaled Khalifa. "Extendable generic base verification architecture for flash memory controllers based on UVM", 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2017 Publication	1%
14	Submitted to Middlesex University Student Paper	<1%

15	www.ijisme.org Internet Source	<1%
16	www.firewallhardware.it Internet Source	<1%
17	Bergeron. "Verification Technologies", Writing Testbenches using System Verilog, 2006 Publication	<1%
18	Submitted to CSU, San Jose State University Student Paper	<1%
19	Submitted to Institute of Technology, Nirma University Student Paper	<1%
20	Submitted to National Institute of Technology Warangal Student Paper	<1%
21	Sigal Asaf, Eitan Marcus, Avi Ziv. "Defining coverage views to improve functional coverage analysis", Proceedings of the 41st annual conference on Design automation - DAC '04, 2004 Publication	<1%
22	Alfonso Martinez Cruz, Ricardo Barron Fernandez, Heron Molina Lozano. "Automated Functional Coverage for a Digital System Based on a Binary Differential Evolution Algorithm",	<1%