Test Content Development For Next Generation Graphics

Major Project Report

Intel Technologies India Pvt. Ltd.

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Mrunmayee Vijay Joshi
(13MECV08)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
June 2015

Test Content Development for Next Generation Graphics

Major Project Report

Phase- I

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology in Electronics & Communication Engineering (VLSI Design)

By

Mrunmayee Joshi

(13MECV08)

Under the guidance of

External Project Guide	Internal Project Guide
Mrs. Kavitha Seshadri	**Dr. N. M. Devashrayee**
Project Manager,	PG Co-Ordinator (VLSI Design),
Intel India Technology Pvt. Ltd.,	Institute of Technology,
Bangalore.	Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
June 2015

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.
2. Due acknowledgment has been made in the text to all other material used.

- Mrunmayee Vijay Joshi
13MECV08

Intel Technology India Pvt. Ltd.

Certificate

This is to certify that the Major Project entitled **“Test Content Development for Next Generation Graphics ”** submitted by **Mrunmayee V. Joshi (13MECV08)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date :
Place : Bengaluru

Kavitha Seshadri,
Engineering Manager,
Intel Corporation Pvt. Ltd,
Bengaluru.

Certificate

This is to certify that the Major Project entitled **"Test Content Development For Next Generation Graphics"** submitted by **Mrunmayee V. Joshi (13MECV08)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Internal Guide	**External Guide**
Dr. N. M. Devashrayee	**Mrs. Kavitha Seshadri**
(Professor,EC)	(Project Manager)
HOD	**Director**
Dr.P.N.Tekwani	**Dr. Ketan Kotecha**
(Professor,EE)	(Director, IT-NU)
Date:	**Place:Ahmedabad**

Acknowledgements

I take this opportunity to thank everyone at **Nirma University and Intel Technologies India Private Limited** for providing me the opportunity to carry out this training work

I would like to express my sincere gratitude to **Dr. Ketan Kotecha** (Director, Nirma University, Ahmedabad) for his continuous guidance, support and enthusiasm. I would take this opportunity to thank **Dr. P. N. Tekwani** (Head of Department, Electrical Engineering),**Dr. D. K. Kothari** HOD,EC Department, Institute of Technology, Nirma University for his constant support and encouragement, for providing all essential resource

I would like to thank **Dr. Niranjan M. Devashrayee** Co-ordinator of VLSI, Nirma Institute of Technology, for his constant guidance and valuable suggestions which were vital to carry out this work

I would like to express my deepest gratitude to my manager at Intel, **Mrs. Kavitha Seshadri** for providing me an opportunity to learn, and also for her mentoring and encouragement during the course of the project work. I owe a great sense of gratitude to whole team for their valuable time in helping me understand the environment and for sharing their knowledge and experience which helped immensely in carrying out this training work.

Lastly, I thank almighty and my parents for their constant encouragement without which this work would not be possible
I wish to thank my friends of my class for their delightful company which kept me in good humor throughout the journey.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

-Mrunmayee Vijay Joshi
13MECV08

Abstract

In today‘s speedily growing world of VLSI circuits, test quality has significant effect on the quality of the product. The increase in complexity of VLSI circuit has made testing more tedious and tough. So devising good tests is one of the most important steps in manufacturing quality microcircuit. Higher quality tests enable screening of chips and also discover defective chips before they leave the manufacturing plant .The quality of the test is represented by its fault coverage through the fault simulation process. The most effective way to test a circuit is to observe its behavior by building a logic model and then using these logic models to check the physical character of the circuit. These logic models are called as the fault models. High testing quality minimizes DPM (Defective parts per million) and thus can significantly reduce manufacturing costs and the probability of defective chips shipping out.

The graphics processing unit (GPU) has become an integral part of today's mainstream computing systems. In multimedia technology, Graphics is the key element. So it should be made sure that it is fault free. There are many different ways to make it fault free mentioned in this report. This project work aims to come up with different methodology to develop test content for Intel's Graphics Processor Unit (GPU) that will meet the fault coverage target.

Contents

Declaration **iii**

Certificate **v**

Acknowledgements **vi**

Abstract **vii**

List of Figures **xii**

1 Introduction **1**

1.1 Today's Scenario 1

1.2 objective 2

1.3 organization of project thesis 2

2 Literature Survey **3**

2.1 GPU Introduction 3

2.2 GPU v/s CPU 4

2.3 GPU Block Diagram 5

2.4 What is Graphics? 6

2.5 2D v/s 3D Graphics 6

2.6 3D Graphics and APIs 7

2.6.1 Microsoft DirectX11 7

2.6.2 OpenGL 8

2.7 DirectX v/s OpenGL 8

2.8 3D Pipeline 9

2.9 High Volume Manufacturing Flow 11

2.9.1 Wafer Sort 11

2.9.2 Assembly 12

2.9.3 Burn-in 12

2.9.4 Post Burn-in check(PBIC) 12

2.9.5 PC Platform Validation (PPV) 12

2.9.6 Final Quality Assurance (FQA) 12
2.10 Summary 13

3 Test Related Concepts 14
3.1 Fault 14
3.2 Fault Modelling 14
3.3 Fault Classification 17
3.4 Fault Equivalence 17
3.5 Fault Collapsing 17
3.6 Fault Dominance 20
3.7 Concept of Fault Simulation 21
3.7.1 Serial Fault Simulation 22
3.7.2 Parallel Fault Simulation 22
3.7.3 Concurrent Fault Simulation 22
3.8 Fault Sampling 23
3.9 Fault Coverage 23
3.10 Detecting Faults 24
3.11 What is Test? 25
3.12 Testing vs Verification 25
3.12.1 Eliminating these Flaws 26
3.13 Structural v/s Functional Testing 27
3.13.1 Functional Testing 27
3.13.2 Structural Testing 27
3.14 Summary 28

4 Fault Coverage Escalating Methodology 29
4.1 Fault Grading 29
4.2 Model Building 30
4.3 Logic Simulation or Logic Validation 30
4.4 Exclusion 30
4.4.1 Manual Exclusion 30
4.4.2 DFT Signal Constraints 31
4.4.3 BIST exclusion 31
4.5 Execute Tests/Faults Simulation 31
4.6 Grading and Evaluation 32
4.7 NOA (Node Observability Architecture) 32
4.8 Summary 34

5 Tools and Technology **35**
5.1 Regression 35
5.1.1 Run command 35
5.2 Test Debugging 36
5.2.1 Checker Error 36
5.2.2 Run Limit Error 37
5.2.3 Test Programming Error 37
5.3 Test_Status 37
5.4 Fixing NOA 38
5.5 Summary 39

6 Results and Analysis **40**
6.1 Fault Coverage 40
6.1.1 Sampler Cluster 40
6.1.2 Pipeline Cluster 41
6.1.3 Interface Unit 42

7 Conclusion and Future goal **43**
7.1 Conclusion 43
7.2 Future Goal 43

APPENDIX **44**

Bibilography **45**

List of Figures

2.1 GPU Block Diagram 5
2.2 2D v/s 3D image comparison 7
2.3 3D-Pipeline Flow 9
2.4 Tesellation of Sphere 10
2.5 High Volume Manufacturing Flow 11

3.1 Bridging Fault 15
3.2 pmos is stuck_short 16
3.3 pmos is stuck_open 16
3.4 Path Delay Fault 17
3.5 Basic gate fault equivalence 18
3.6 Exapmle for Fault equivalence 18
3.7 Finding Fault equivalence 19
3.8 Reduced Faults 19
3.9 Circuit after the fault equivalence 20
3.10 Reduced Fault Dominance 20
3.11 Concept of Fault Simulation 21
3.12 Testing the block by giving a test-pattern 24
3.13 Fault Detected using test-pattern 25
3.14 Testing vs Verification 26
3.15 structural and functional fault testing 28

4.1 Basic Fault Grading Flow 29
4.2 Fault Simulation Process 32
4.3 NOA register 33
4.4 Region Covered by simple tests and NOA tests 34

5.1 Regression Status in NetBatch Flow Manager 36
5.2 Regression Status in NetBatch Flow Manager 37
5.3 Regression Status in NetBatch Flow Manager 38

6.1 Sampler Cluster Fault Coverage Chart 40
6.2 Pipeline Cluster Fault Coverage Chart 41

6.3 Interface Unit Fault Coverage Chart 42

Chapter 1

Introduction

1.1 Today's Scenario

For Integrated Circuits the most commonly used manufacturing technique is High Volume Manufacturing. In order to detect the faults which are induced during manufacturing, good quality tests are required. So developing good tests is one of the most important steps in manufacturing quality microcircuits. The increase in complexity of design results in increasing effort and time to develop high quality tests. Even though the complexity increases, the time lines for the project in most cases decreases. Thus there is a need to use advanced methodologies and flows that look to decrease the time and effort.
Manufacturers measure product quality by the number of rejects or the number of defective parts per million shipped. The quality of the test is represented by its fault coverage through the fault simulation process. However, high fault coverage does not guarantee that the test is capable of detecting all manufacturing faults. The high fault coverage simply means that a given test is good enough to detect all or most of the faults considered. In other words, a test with 100 fault coverage may still fail to detect faults outside the considered fault model. Hence its important to come up with a good methodology and good set of fault models to generate a good test. This model should be able to represent the most common type of fault such as stuck(0) and stuck(1) during manufacturing. Usually fault models are built for a RTL design and the tests are executed on these models to determine the faults detected by the test. Fault grading is a common methodology used to examine the efficiency of tests in determining such kind of faults

1.2 objective

The main objective of this project is to develop test cases compatible environment for the new generation graphics ,that will enable us to detect maximum faults in the different units of the design and escalate the over all fault coverage. These test cases will help us for the post silicon validation process to test the manufactured chips. We wish to reach the over all fault coverage of 85% by using various fault analysis methodology.

1.3 organization of project thesis

- The first chapter is the brief introduction of the main area of the project, discuss the present scenario, objective of the work and report organization
- The second chapter is the Litreture Survey of all the concepts which were required to study during this project. Graphics, GPU, Graphics APIs, 3D pipeline, High Volume Manufacturing Flow
- The Third chapter is the basic concepts related to Testing such as Fault Modeling, Fault Classification, Structural vs functional Testing, Fault Simulation, Fault Coverage and the concept of observable and controllable points.
- The fourth chapter consists of the basic flow of Fault Grading process.
- The Fifth chapter is the work done in the whole project using different tools and technologies.
- Sixth chapter contains results of all the work done during the project.
- The last chapter consist of the conclusion of the project and the future scope.

Chapter 2

Literature Survey

2.1 GPU Introduction

Graphics Processing unit is also called **Visual Processing Unit** (VPU). They are mostly used in mobile phones, personal Computers, game consoles, workstation etc. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. In a personal computer, a GPU is present on a video card, or can be on the motherboard, or in certain CPUs, on the CPU die. More than 90% of new desktop and notebook computers have integrated GPUs, which are usually less powerful than those on a dedicated video card.

The building block of every computer image is called a picture element or pixel. Three color components - red, green and blue make a pixel. The processor computes these color values for each pixel on the screen and then stores them in the form of numbers in a special type of local memory. This type of Random Access Memory(RAM) is used to store video information because it connects to both the Central Processing Unit and the graphics controller.

The information in the local memory is pushed to the graphics controller when it is needed. The graphics controller takes this information and convert it into a wave signal that will determine the voltage of three electron guns. The electron guns are grouped in three and each gun is responsible for a single color component. The intensity of each is determined by the voltage of the incoming signal. The electron beams are shot at the back of the screen which is coated with dots of chemicals called phosphors. A pixel is made up of three phosphors, one chemically constructed to glow red, one green, and one blue. These three phosphors that make up a pixel are often called a phosphor triad. By varying the intensities of the three color components almost any color can be created depending on the size of the smallest variation.

The GPU can be used in following forms:

Dedicated graphics card:This kind of GPUs are the most powerful class. They interface with the motherboard by means of an expansion slot such as PCI Express (PCIe) or Accelerated Graphics Port (AGP) and can usually be replaced or upgraded with relative ease. The term "dedicated" does not means that most dedicated GPUs are removable. A dedicated or discrete GPU has its own independent source of video memory, leaving the RAM of the system untouched.
Integrated graphics card:An integrated graphics processing unit (GPU) doesn't use its own RAM, but utilizes the system's memory instead. The video card can use anywhere between one and five percent of the available memory for graphics processing. This percentage of memory utilization varies depending on the size of task, especially in case of multitasking or playing a game. The integrated unit is that it is cheaper, which in turn means a less expensive computer. An integrated graphics card also generates much less heat than a dedicated video card and drastically less power consumption, which improves the overall battery life.

In Integrated graphics card, GPU resides on the same chip die as of CPU. It communicates with the CPU via on-chip bus. The GPU is controlled by CPU through a direct interface of memory mapped I/O registers and indirectly by parsing commands that CPU places in the memory. The Display interface and Block image transferrer are controlled primarily by direct CPU register addresses, while the 3D pipeline and Media pipelines and the parallel Video Codec Engine are controlled primarily through instruction lists in memory.

The subsystem consists of array of Execution Units and set of shared functions. It can be accessed by media and 3D pipelines. Shared functions are hardware units which serve to provide additional functionalities to EUs. It is implemented when the demand for a particular function does not justifies the cost of per-EU. So, a single entity is implemented outside the Execution Unit and is shared among them

2.2 GPU v/s CPU

The CPU (central processing unit) has often been called the brains of the PC. But that brain is being more enhanced by another part of the PC, the GPU (graphics processing unit), which is the soul.
A GPU is tailored for highly parallel operation while a CPU executes programs serially. For this reason, GPUs have many parallel execution units and higher transistor counts, while CPUs have few execution units and higher clock speed

Architecturally, the CPU is composed of only few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously
Modern GPUs are very efficient at manipulating computer graphics. Their are highly parallel in structure which makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel

2.3 GPU Block Diagram

3D graphics creates the illusion of solid objects in a real environment
This is the Block diagram of GT2. Here GT2 stands for graphics with two slices (Slice-1 and Slice-2).
The GPU can consist of one,two, three or four slices, depending on requirement of speed

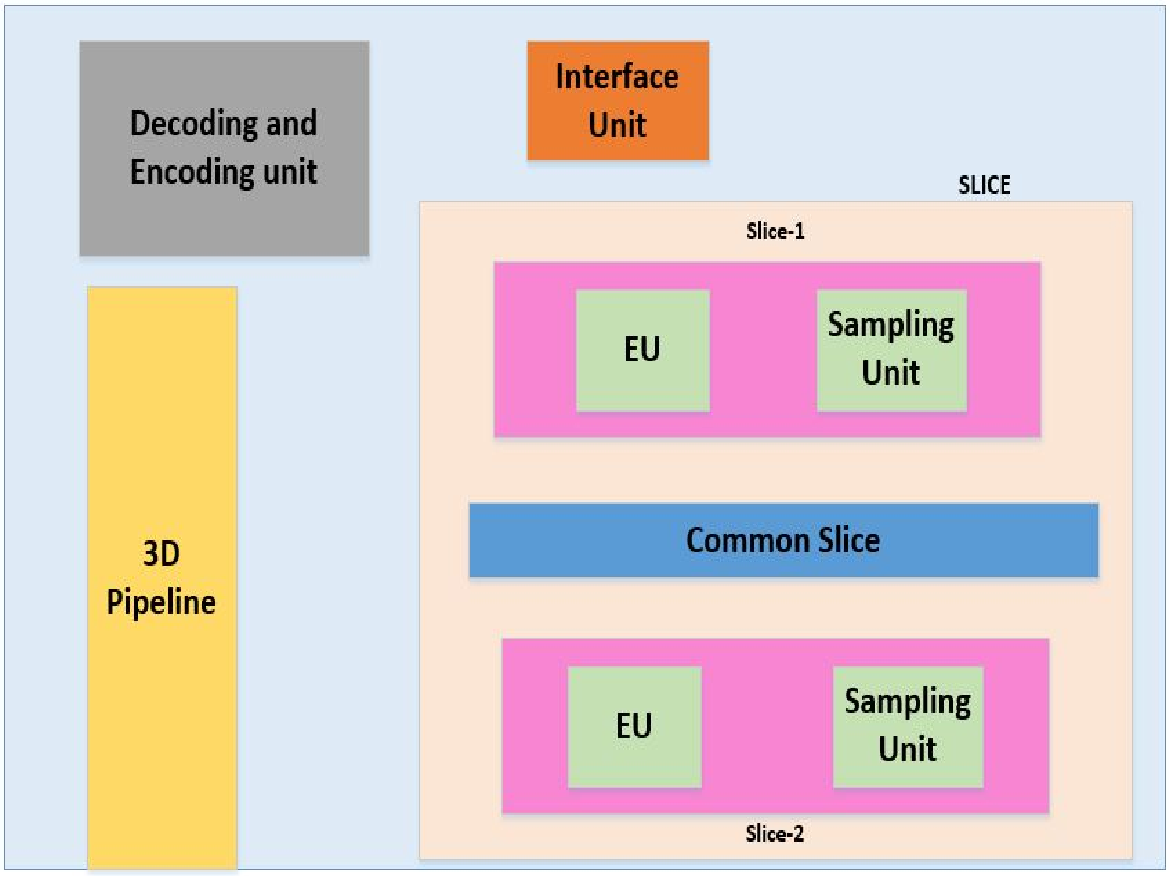


Figure 2.1: GPU Block Diagram

Slice-1, Slice-2 and common slicing togather are conside as slice. 3D Pipeline, Interface unit and Decoder and encoder unit are consider to be unslice.

- **GTI:**
  It is an interface between GPU and other peripheral devices. It helps GPU to interface with the outside world. Physical Address mapping is the main target.

- **Sampler:**
  It does texture mapping, shadowing. It also help in increasing and decreasing the size of picture.

- **Slice common:**
  This Unit of GPU divides the captured image into many triangles. It decides at what angle should the triangle be.

- **Decoder and Encoder:**
  This basically decides what should be the format of the file that is to be sent out.

- **EU:**
  It is called Execution unit. It is a part that performs the operations and calculations called for. It may have its own internal control sequence unit like, some registers, and other internal units. It helps in doing parallel computing.
  The skylake processors

GT1	12 execution unit
GT1.5	18 execution unit
GT2	24 execution unit
GT3	48 execution unit

- **3D Pipeline:**
  This is the main part of graphics processor. It renders the images

2.4 What is Graphics?

Human mind always thinks in terms of pictures. It makes the understanding easy. So the world of graphics came into existence in reality. To operate the graphics hardware there are some APIs in the market, such as, DirectX, OpenGL, etc. APIs are an interface that allows a programmer to render graphics more easily

2.5 2D v/s 3D Graphics

The fig 3.1 is a very simple example of the difference between 2-D and 3-D images. A 3-D image shows much more realism and depth than a 2-D image. Some of the details of real life that makes 3-D images real includes numerous shades of color, lighting effects and textures. Though 2-D images can mimic the realism effects of 3-D, they cannot duplicate them. Every piece of a 3-D image that is stored allows one to view it accurately from any sides or angle. In the case of a 2-D picture that views is limited to a single point of view

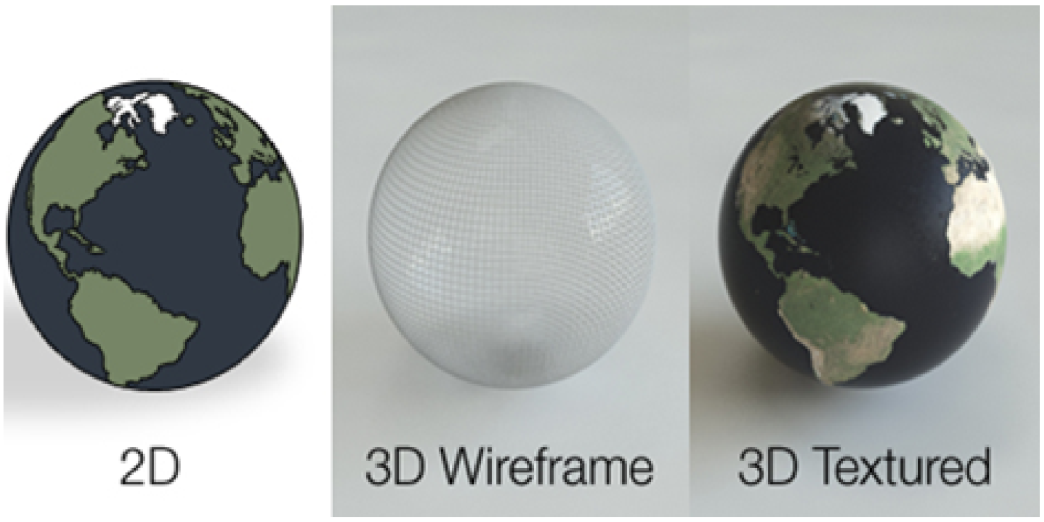


Figure 2.2: 2D v/s 3D image comparison

2.6 3D Graphics and APIs

3D computer graphics is the science, study, and method of projecting a mathematical representation of 3D objects onto a 2D image using visual tricks such as perspective and shading to simulate the eye's perception of those objects. Some of the 3D APIs are:

2.6.1 Microsoft DirectX11

Microsoft DirectX is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially programming games on video, on Microsoft platforms. Originally, the names of all these APIs began with "Direct", such as Direct 3D, DirectDraw, Direct Music, Direct Play, Direct sound and so forth. The name DirectX was coined as shorthand term for all of these APIs (the X standing in for the particular API names) and soon became the name of the collection. As Direct3D is the most widely publicized component of DirectX, it is common to see the names "DirectX" and "Direct3D" used interchangeably. Direct3D 9Ex, Direct3D 10 and Direct3D 11 are only available for Windows VistaÂ® and Windows 7Â®. Each of these new versions was built to depend upon the new windows display driver model that was introduced for Windows Vista.

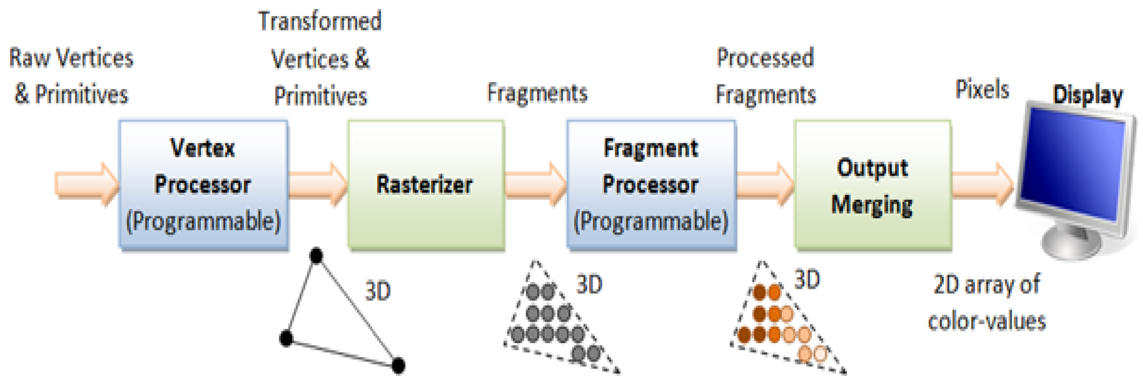
2.6.2 OpenGL

OpenGL stands for Open Graphics Library, is a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D vector graphics. It is an alternative for DirectX. It is a specification for writing applications that produce 2D as well as 3D computer graphics. The interface consists of 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. It is managed by the non-profit technology consortium Khronos Group.

2.7 DirectX v/s OpenGL

DiretX	**OpenGL**
Supported on Windows	Supported on Windows, Mac, Linux and several other Unix variants
Object Oriented interface	Function Oriented interface
Updates very frequently(once in a year) and supports the latest card features.	Takes some time to reach a new standard
DirectX contains tools to deal with such components of a game as sound, music, input, networking, and multimedia	OpenGL is strictly a graphics API

2.8 3D Pipeline


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
Fragments
Fragment Processor (Programmable)
Processed Fragments
Output Merging
Pixels
Display
3D
3D
3D
2D array of color-values

3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (*x*, *y*, *z*) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

Figure 2.3: 3D-Pipeline Flow

1. From the input scene, a set of primitives (Commonly used primitives are triangles, lines, and points) are generated by the application software, each of which has vertices (A ”corner” of a primitive, having a position in space and other attributes like color, texture coordinates describing it). The host interface which is the communication bridge between the CPU and the GPU, receives commands from the CPU and pulls geometry information from main memory.

2. For each vertex input, Vertex Shader unit generates one vertex output after applying some arbitrary operations on vertices attributes. Eg. Like skinning, lighting etc. such transformation operations are used.No new vertices are created in this stage, and no vertices are discarded.

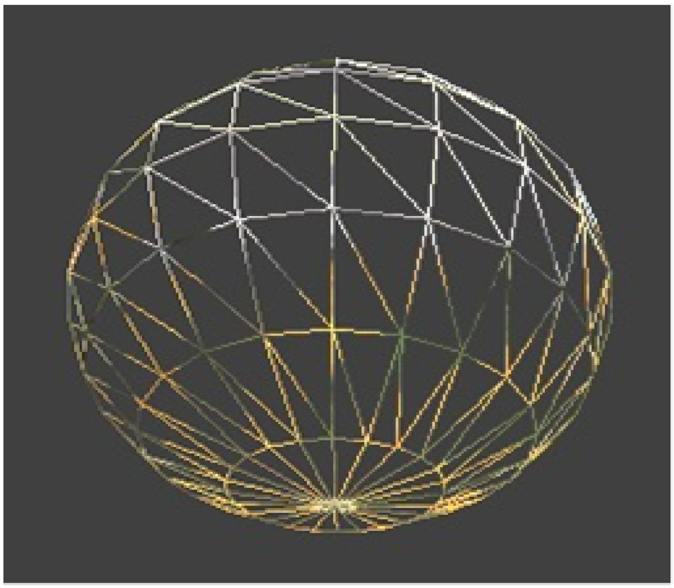


Figure 2.4: Tesellation of Sphere

3. The Rasterizing unit is responsible for displaying 3-D shapes on the computer. The vector graphics format is converted to raster image (dots or pixels) format. Convert each primitive (connected vertices) into a set of fragments. A fragment is treated as a pixel in 3D spaces, which is aligned with the pixel grid, with attributes such as position, color, normal and texture. A fragment is generated if and only if its center is inside the triangle. A fragment is 3-dimensional, which has (x, y, z) coordinates. The (x, y) are aligned with the 2D pixel-grid. The z-value (not grid-aligned) denotes its depth.

4. Each fragment is fed into fragment processing unit as a set of attributes (position, normal, texture coordinates etc), which are used to compute the final color for this pixel The computations taking place here include texture mapping and math operations.

5. The Out Merging Unit combines the fragments of all the primitives (in 3D space) into 2D color-pixel for the display.

In modern GPUs, the vertex processing stage and fragment processing stage are programmable. To perform your custom transform for vertices and fragments, vertex shader and fragment shader are programmed. These shader programs are written in high level languages such as GLSL (OpenGL Shading Language), HLSL (High-Level Shading Language for Microsoft Direct3D), or Cg (C for Graphics by NVIDIA).

2.9 High Volume Manufacturing Flow

High Volume Manufacturing (HVM) is the mature stage of manufacturing. HVM is usually marked by ongoing test refinements to further reduce test time. The HVM flow is shown below:

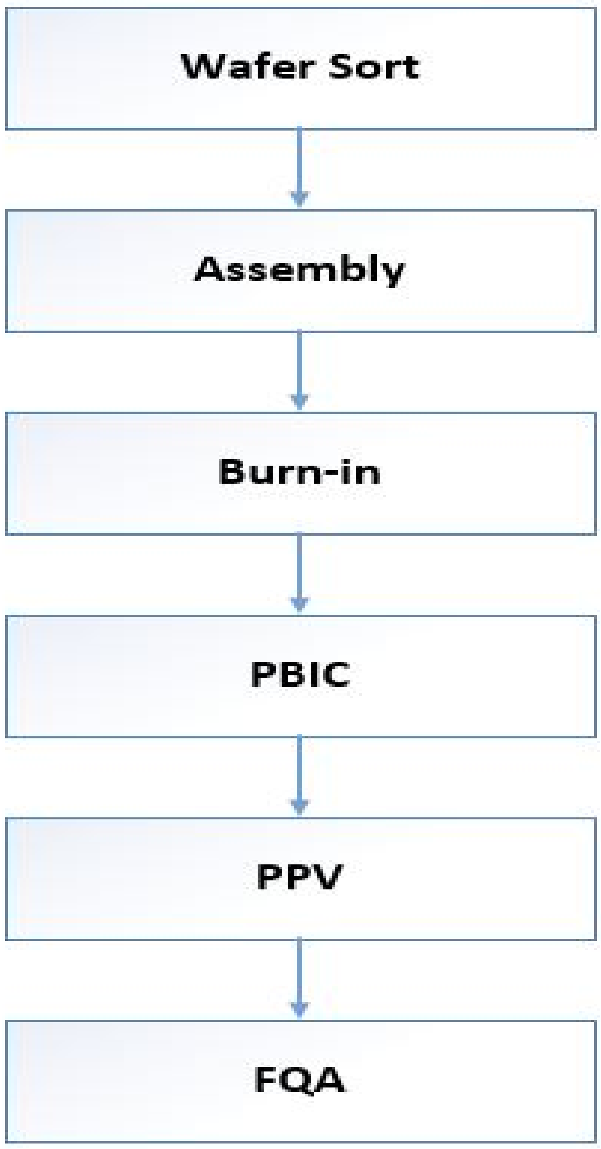


Figure 2.5: High Volume Manufacturing Flow

2.9.1 Wafer Sort

Wafer Sorting is the process where each die on the wafer is tested. Passing dies are then sorted from the failing ones. Only the passing die is assembled into package.

2.9.2 Assembly

In Assembly phase packaging is done.

2.9.3 Burn-in

Burn-in consists of functionally exercising a part at a high temperature and voltage. The primary goal of burn-in is to accelerate particle defects and processing problems to failure. The dominant failure mechanisms expected on the multi-layer metal processes are generally particles which display voltage and/or temperature acceleration.

2.9.4 Post Burn-in check(PBIC)

Units are categorized into the various accept bins at PBIC. Data gathered at this step is used to estimate the overall random defects in the manufacturing process, estimate test costs, track bin splits, and forecast assembly related yields. PBIC is used as the primary vehicle for screening several classes of defects, like
a) Assembly related defects
b) Hot temperature sensitive defects
c) Fab process related defects

2.9.5 PC Platform Validation (PPV)

PPV is also known as PC testing or product platform validation The PPV methodology is intended to provide a methodology to qualify new products, screen products with higher defects per million (DPM), monitor low DPM products, and set criteria for going from one PPV mode to the other (from screen to monitor, and from monitor back to screen); measure and validate the target product DPM against the corporate goal.

2.9.6 Final Quality Assurance (FQA)

FQA is the last stage and occurs in all the manufacturing flows. The purpose of FQA is to provide a final check of outgoing electrical quality. Failures that are found at FQA are often caused by:
a) Units that have been damaged physically or electrically during the manufacturing flow
b) Accidental mixing of failing units with passing units in production.

2.10 Summary

GPU is the basic requirement for the 3D worl. 3D graphics is creating illusion of the 2D object as 3D object, using coloring and patterning techniques to make it look real.
There are different APIs which helps to use GPU. There are mainly two APIs DirectX and OpenGL using which we design pipeline. 3D pipeline is the main functioning block.
High Volume Manufacturing is the technology used to manufacture chips. It goes through many test process. The main difference between Testing and Verification is that Verification is done pre-silicon while Testing is done post-silicon.

Chapter 3

Test Related Concepts

A chip may fail due to many reasons like design or fabrication flaws, environmental factors, or a combination of these. The resulting physical defects consists of break in lines, shorts between lines at the interconnection level, shorts through the insulator separating different levels, shorts to substrate, point detects and imperfections such as scratches across the chip. Testing the actual physical defect is difficult. An effective way to test the circuit is by observing the circuits behavior and building the logic model.

3.1 Fault

Fault is the physical defect in the circuit design. Faults can occur due to some of these failures:

- **Processing Faults**
  It is the extra or missing material ,preliminary caused by dust particles on the mask or wafer surface or in the processing chemicals.
- **Material Defects**
  It is the fault caused by cracks or crystal imperfection. Metal Defects is also caused by ion migration.
- **Time Dependent failure**
  After certain period of operation due to lack of sustainability of material there occurs Dielectric breakdown and electromigration.

3.2 Fault Modelling

A fault is a physical defect in a circuit or a system. It may or may not lead to system failure. Fault Models assist in developing tests. It is important to model faults as it becomes easier to identify target faults. A fault model is an engineering model of something that could go wrong in the construction or operation of a piece of equipment. From the model, the designer or user can then predict the consequences of this particular fault. The different fault models in digital circuits include:

- **Stuck-At faults** can be Single Stuck-At or Multiple Stuck-At fault. The signal here is stuck at 0 or 1.In single stuck-@ model, it is assumed that only one line is faulty. The faulty line is permanently set to 0 or 1. This fault can be at input or output of a gate. There are many advantages of using this model. The complexity is greatly reduced. Many physical defects can be modelled by same logical single stuck-at faults. This model is technology independent in the sense of using TTL, ECL, CMOS, etc. Single stuck-at tests cover a large percentage of multiple stuck-at faults.
  Stuck-at model is a successful model, as it assist to cover high percentage of detects, and hence it is widely used. Fault listing models real manufacturing defects to faults that can be simulated, analyzed and tested.

- **Bridging fault** occurs when two or more distinct lines are shorted togather. Depending on the circuitry it may result as wired-and or wired-or

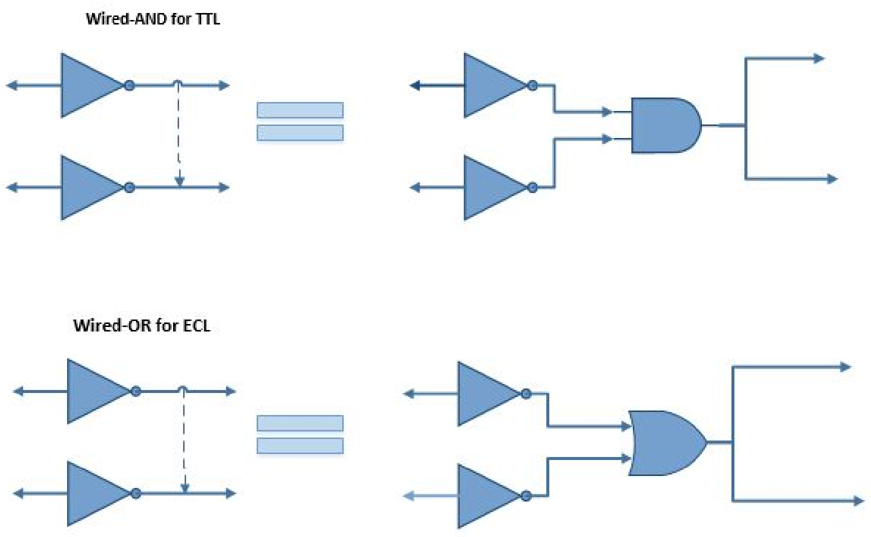


Figure 3.1: Bridging Fault

- **Transistor Fault** are the faults at transistor level for CMOS logic gates. Transistor may be stuck-open or stuck-short.

  In stuck-short, the transistor always conducts.

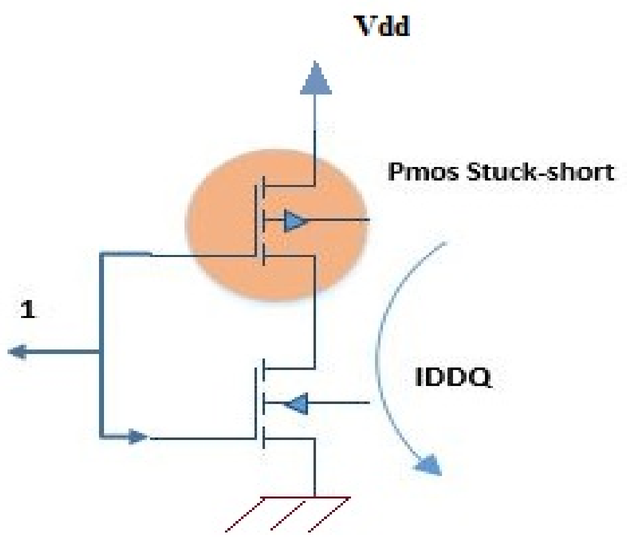


Figure 3.2: pmos is stuck_short

Here the Pmos is stuck-on, so it will always conduct irrespective of any inputs to gate. When the input to an inverter is 1 then Nmos and Pmos will be ON simultaneously and a large quiescent current will flow called IDDQ.Detection of a stuck-short fault requires the measurement of quiescent current (IDDQ)
In stuck-open

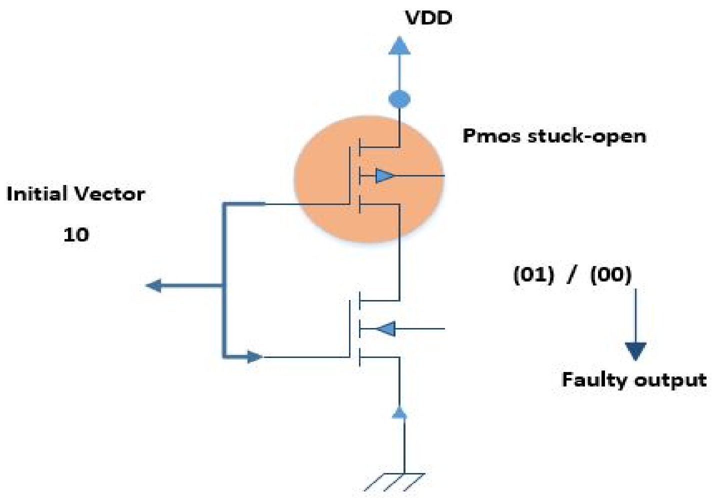


Figure 3.3: pmos is stuck_open

Detection of a stuck-open fault requires two vectors

- **Delay faults**,where the signal eventually assumes the correct value, but more slowly (or rarely, more quickly) than normal. A defect can affect the speed of the path in the circuit.

Figure 3.4: Path Delay Fault

3.3 Fault Classification

Faults are classified into following categories
Detected [DET]: A value opposite to the good machine value has been observed.
Undetected [UND]: Good and faulty machines have the same value.
Oscillatory [OSC]: The faulty machine oscillates (does not settle in the specified time).
Untestable [UNT]: Fault effect cannot be observed as the fault is tied, blocked, redundant, or propagates only an X value.

3.4 Fault Equivalence

When same number of tests, detect two faults 'f1' and 'f2' then f1 and f2 are called equivalent faults. Two faults are detected by same tests then those faults are functionally identical. The fig 3.5 shows the fault equivalence for the basic gates. In all the basic gates there are some faults which have equivalence.

3.5 Fault Collapsing

Every single fault in the logic circuit, can be divided into disjoint equivalence subsets,where all faults in a subset are mutually equivalent. A collapsed fault set contains one fault from each equivalence subset.

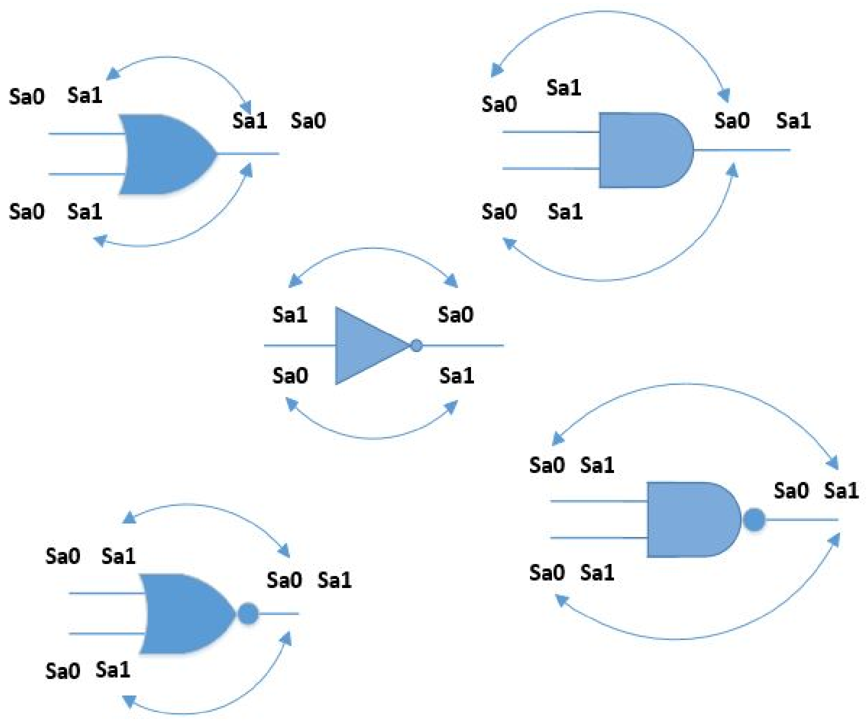


Figure 3.5: Basic gate fault equivalence

Below is the example of finding Fault equivalence to reduce fault list.

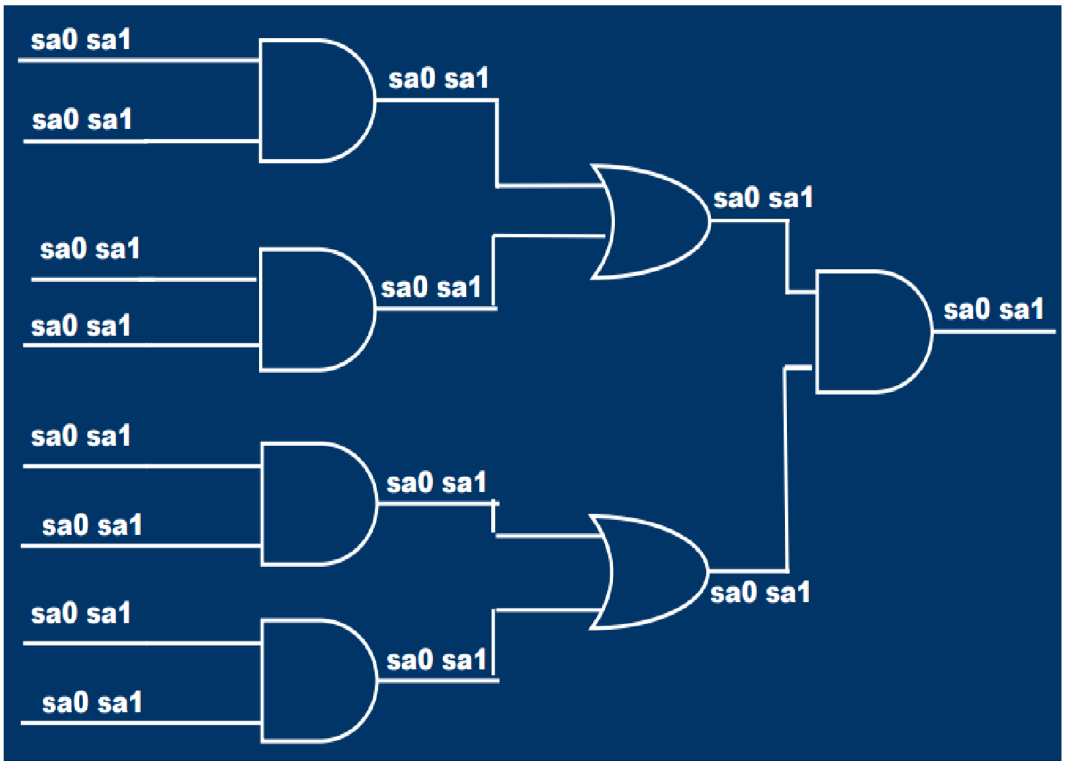


Figure 3.6: Exapmle for Fault equivalence

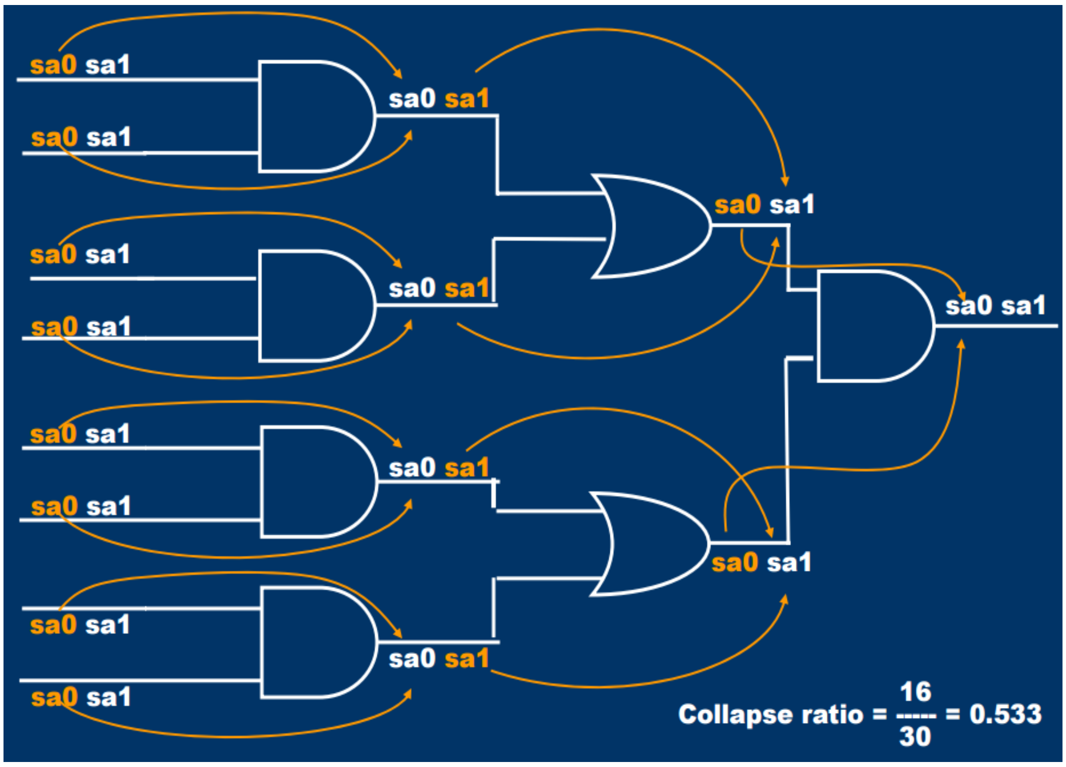


Figure 3.7: Finding Fault equivalence

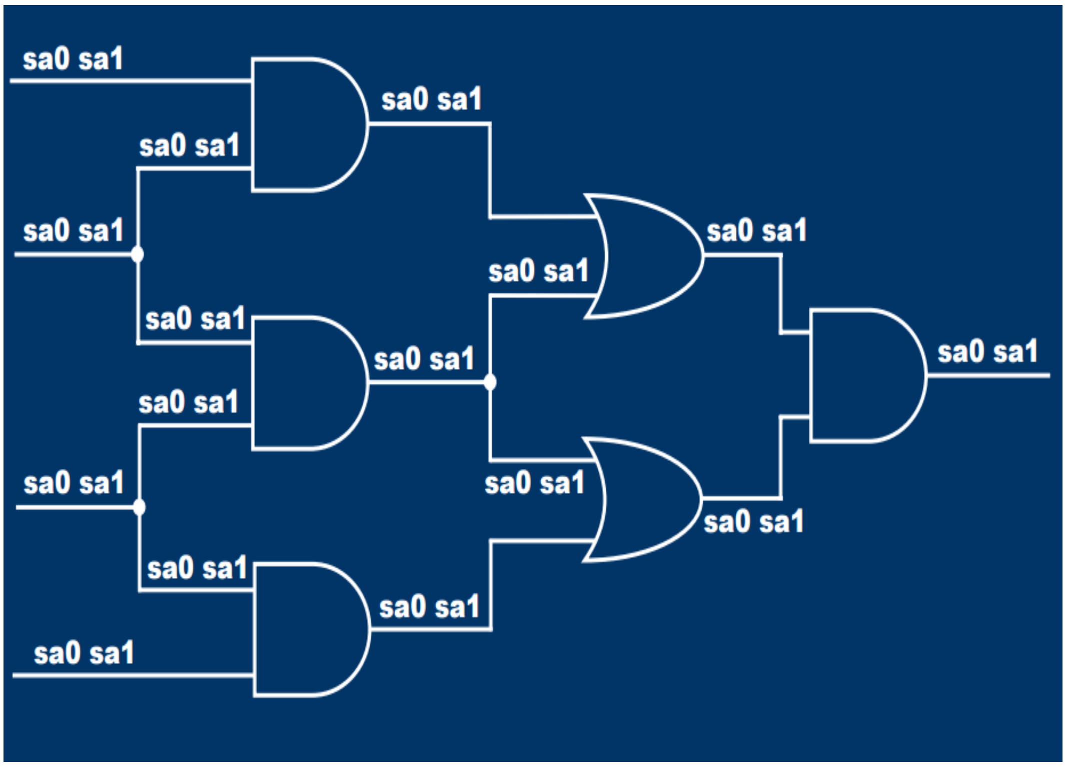


Figure 3.8: Reduced Faults

3.6 Fault Dominance

If some tests of Fault 'f1' are able to detect another fault 'f2' then fault 'f2' is said to be dominating fault 'f1'. So we remove dominating fault. This method of removing dominating faults is called **Dominance Faults Collapsing** Below is the example of Dominance fault.

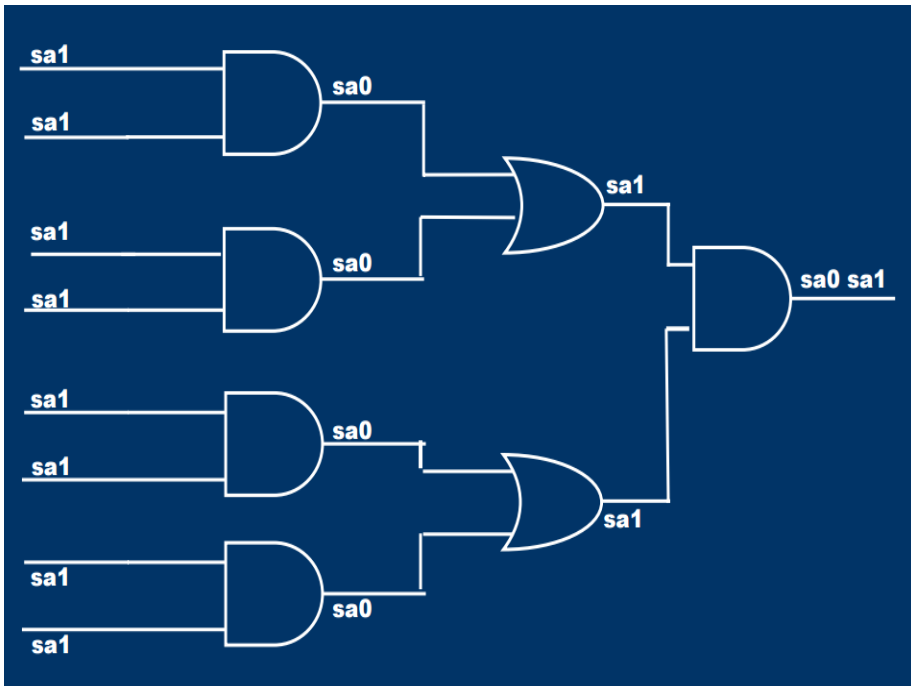


Figure 3.9: Circuit after the fault equivalence

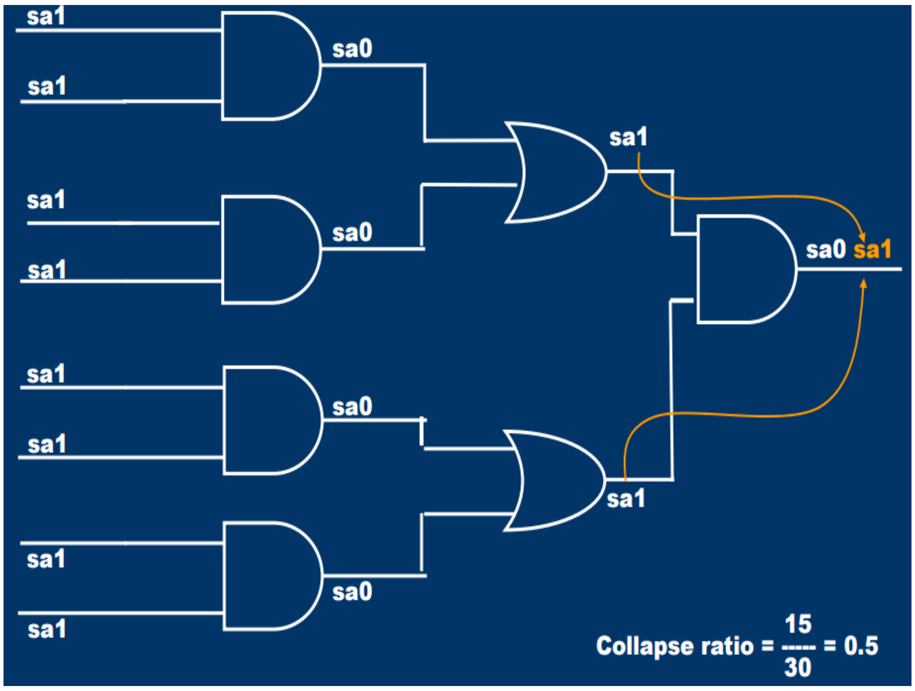


Figure 3.10: Reduced Fault Dominance

3.7 Concept of Fault Simulation

The major step in fault grading is fault simulation. Fault simulation is essential tool for test development. In this method faults are introduced in the defect free model of the circuit and then the behavior of the circuit is observed.

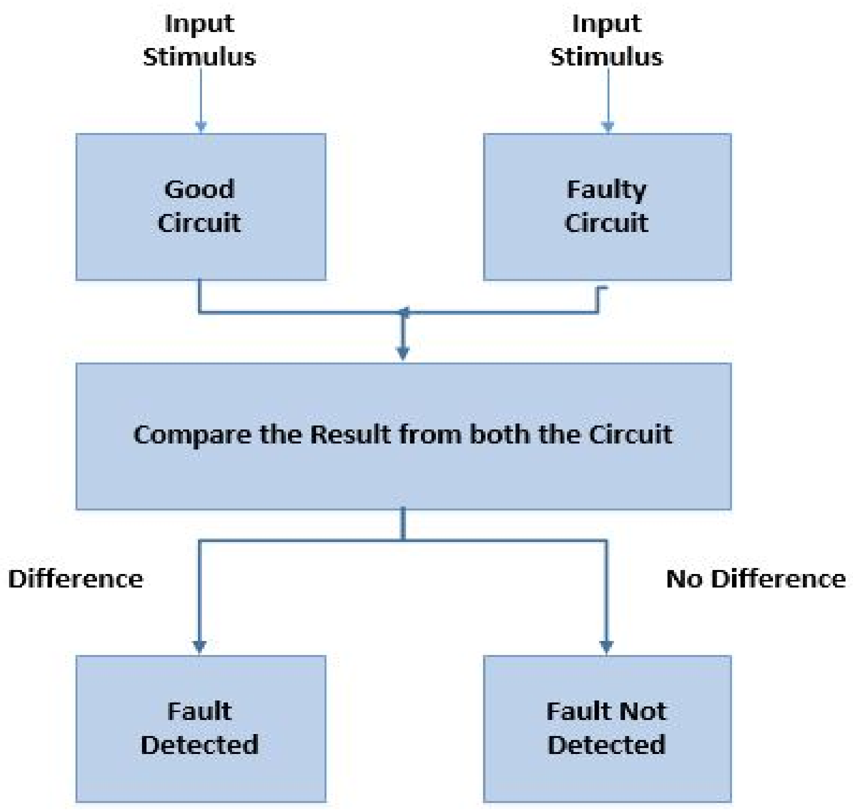


Figure 3.11: Concept of Fault Simulation

The test vectors are used to detect the difference between the two circuits. If a logic discrepancy is observed between the output of the defect free circuit and the faulty circuit, the fault is said to be detected by the test. The tests written for detecting the faults are simulated by the fault simulation tool on the fault model to find out which all faults are detected. There are nodes in RTL design which allow the tool to inject the test vector and observe the propagated fault, known as control point and observation point. The fault simulation tool will compare the outputs of the good model and the faulty model to check whether the test vector is able to detect the fault or not. These controllable and observable points are introduced in a design as scan-in flops and scan-out flops (DFT circuit). Some of the fault simulation algorithm are as follows:

3.7.1 Serial Fault Simulation

Serial fault simulation is the simplest fault-simulation algorithm to implement. It simulates two full instances of the circuit simultaneously. One instance corresponding to good machine and the other corresponding to the faulty machine.

- First Simulate the fault-free circuit and save the response.
- Modify netlist by injecting one fault
- Simulate modified netlist, vector by vector, comparing responses with the saved responses
- If response differs, report fault detection and suspend fault simulation of remaining vectors.
  From step two to last all the steps should be performed for all the faults.

3.7.2 Parallel Fault Simulation

To improve simulation time, parallel fault simulation can be used which simulates several full instances of the circuit (each one represents a different faulty machine with one fault each) simultaneously with the good machine.

3.7.3 Concurrent Fault Simulation

Concurrent fault simulation is the most widely used fault-simulation algorithm and takes advantage of the fact that a fault does not affect the whole circuit. So there is no need to simulate the whole circuit for each new fault. In concurrent simulation the good circuit is simulated completely. Then inject a fault and re-simulate a copy of only that part of the circuit that behaves differently (this is the diverged circuit). For example, if the fault is in an inverter that is at a primary output, only the inverter needs to be simulated—one can remove everything preceding the inverter.
Keeping track of exactly which parts of the circuit need to be diverged for each new fault is complicated, but the savings in memory and processing that result allow hundreds of faults to be simulated concurrently. Concurrent simulation is split into several chunks; one can usually control how many faults (usually around 100) are simulated in each chunk or pass. Each pass thus consists of a series of test cycles. Every circuit has a unique fault-activity signature that governs the divergence that occurs with different test vectors. Thus every circuit has a different optimum setting for faults per pass. Too few faults per pass will not use resources efficiently. Too many faults per pass will overflow the memory. So the number of faults per pass should be chosen wisely.

3.8 Fault Sampling

Fault Simulation is usually done on a randomly selected subset (sample) of faults. So the measured coverage in the sample is used to estimate the fault coverage in the entire circuit. Without fault sampling, complexity of fault simulation depends on:

1. Number of gates
2. Number of faults
3. Number of vectors

With Fault Sampling, Complexity of fault simulation depends on:

1. Number of gates
2. Number of vectors

So the dependency of fault simulation process on the number of faults can be removed (reduced to a constant), if fault sampling is done.
For large circuits, the accuracy of random fault sampling only depends on the sample size and not on the circuit size. This method has significant advantages in reducing CPU time and memory needs of the simulator. But it has the disadvantage of limited data on undetected faults.

3.9 Fault Coverage

Fault coverage refers to the percentage of some type of faults that can be detected during the test of a system. High fault coverage is desirable during manufacturing test. Several techniques such as Design For Test (DFT) and automatic test pattern generation are used to increase it.
In digital electronics for example, stuck-at fault coverage is measured by sticking each pin of the hardware model at logic ’0’ and logic ’1’, respectively, and running the test vectors. If at least one of the outputs differs from what is to be expected, the fault is said to be detected.
A fault coverage test passes when at least a specified percentage of all possible faults can be detected. If it does not pass, at least three options are possible. First, the designer can augment or otherwise improve the vector set, perhaps by using a more effective automatic test pattern generation tool. Second, the circuit may be re-defined for better fault detectability (improved controllability and observability). Third, the designer may simply accept the lower coverage.
The complete detection test set is a set of tests that can detect any detectable faults in a class of faults. The quality of a test is measured by fault coverage. For single stuck-at fault

model, the coverage requirement is greater than 95%.
Fault Coverage is given by: Fault Coverage =

$$\frac{DET}{TotalFaults - UNT} \tag{3.1}$$

Where DET = Faults detected by simulation

UNT = Untestable

3.10 Detecting Faults

The tests written to detect the faults are simulated by the fault simulation tool on the fault model to check the number of faults covered. There are some points in the RTL design which allows us to inject and also observe the value, such points are called observable points and controllable points. below example explains the importance of these points.

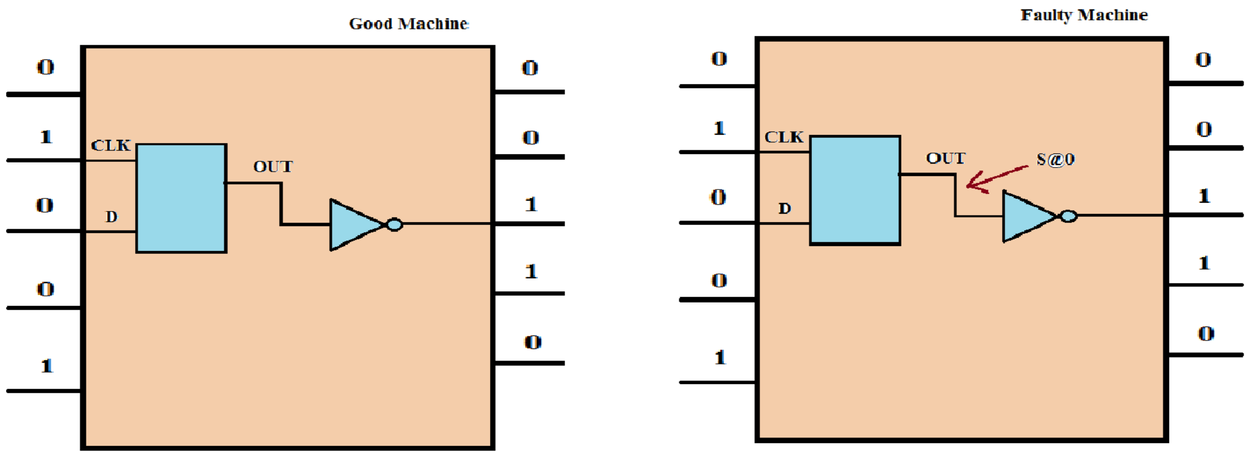


Figure 3.12: Testing the block by giving a test-pattern

The Figure shows the output for the vector 01001. Both the machine i.e., faulty machine and good machine gives the same outputs. So this vector does not help to detect the stuck at fault.

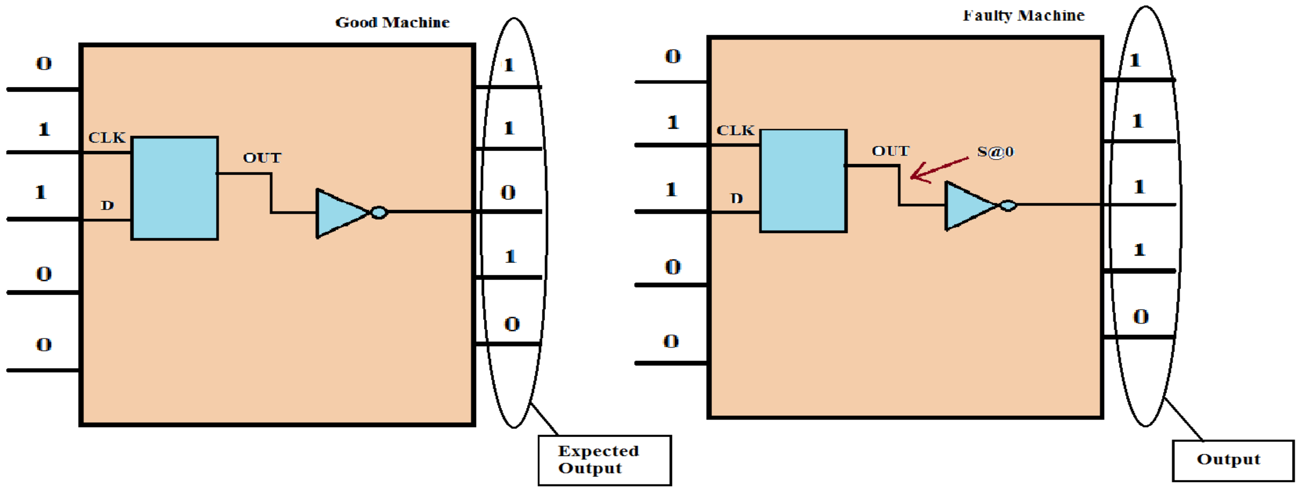


Figure 3.13: Fault Detected using test-pattern

This figure shows the output for the vector 01100. With this input the output of good machine is 11010 and the output of faulty machine is 11110. So by this input vector we can detect the stuck-@ fault

3.11 What is Test?

It is a manufacturing step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect.There are two types of tests:

1. Ideal Tests are the tests which detects all the tests in the manufacturing process. It passes all the functionally good chips and fails all the defective chips.
2. Real tests are the tests which are difficult to generate that detect every possible fault in the chip due to high design complexity.
   Some good chips are rejected such fraction of the chips is called **yield loss**. Some bad chips are shipped, such fraction of chips are called **test escape(defect level)**

3.12 Testing vs Verification

There are software flaws observed at three level:

1. **Defect:**
   A defect in an electrical system is the unintended difference between the implemented hardware and its intended design.

2. **Fault:**
   A representation of a defect at the abstracted level is called a fault.The fault is imperfection in function while the defect is imperfection in hardware.

3. **Error:**
   A wrong output signal produced by a defective system is called an error. An error is an effect whose cause is some defect.

3.12.1 Eliminating these Flaws

These flaws can be eliminated before manufacturing or before shipping the defected chip to the market. That can be done by verification and testing.

- Testing:
  It is running program with set of good inputs to gain the confidence that the software have few defects. In this we make some tests cases, run them and the main purpose is to minimize the failure frequency.

- Verification:
  It verifies the correctness of the design. Verification is carried by various processes like simulation, hardware emulation or formal verification method. It is responsible for quality of design.

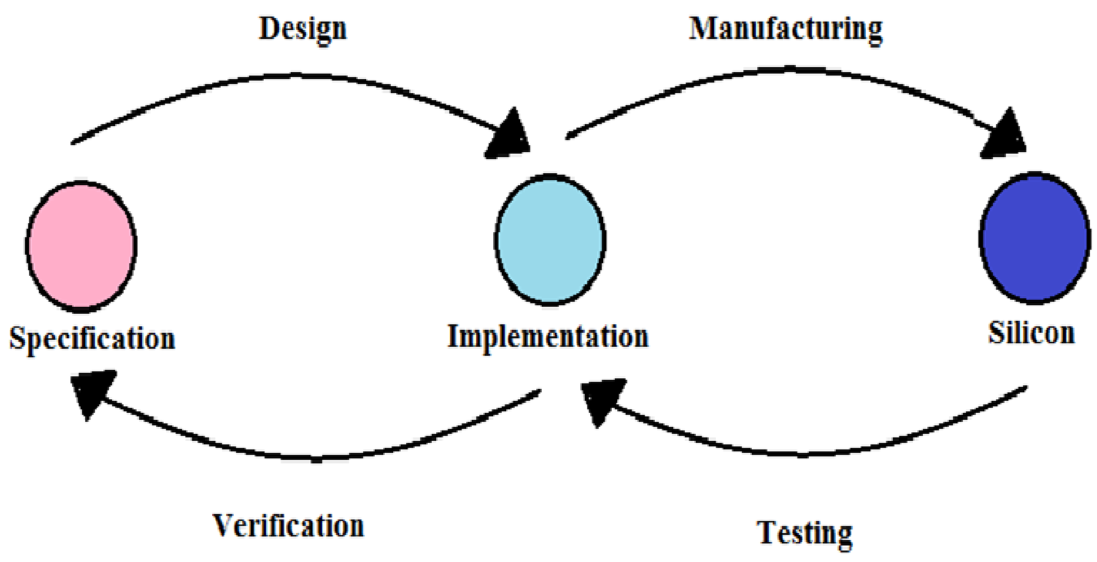


Figure 3.14: Testing vs Verification

Verification is the process carried before manufacturing, while Testing is carried on the manufactured chips. Verification is the software process carried out again and again on the

implemented design before manufacturing is done.
Verification checks "Are we building the Design right?", while Testing checks "Are we building right design". Testing is carried out on the all manufactured chips 'once'. Go/No-go decision is made. If it passes then it is shipped out if failed then it is thrown away, not shipped.Testing mainly consist of two parts:

1. Tests generation: The software process is executed during design. These tests generation should be done carefully as these tests are used to check millions of chips. Once generated tests cannot be changed

2. Tests application: The tests which are generated are used to tests the chips.

3.13 Structural v/s Functional Testing

Testing a digital circuit can be divided into two categories: functional and structural. Functional testing is the testing done to verify the functional operations of the circuit. It confirms that the design compiles to the functional specification of the chip i.e. the implementation should do what it is intended to do. This testing is usually developed in the RTL design stage. Structural testing is the verification for proper construction of each element in the circuit. Fault Grading provides a tool for developing an effective structural testing test suite.

3.13.1 Functional Testing

Functional Testing is called 'black box' testing. In Functional testing exhaustive testing is done. All the combinations of input is checked. For example: Consider number of inputs are 'n', then 2^n combination inputs are required to be given to check its functionally correct or not.

3.13.2 Structural Testing

Structural Testing is called 'white box' testing. In structural Testing, we need to check whether the structure is fault free? for this we require the knowledge of internal logic to develop the test cases. This can be done by considering stuck-at-1 and stuck-at-0 at all the nets. Suppose there are n nets. so there are total 2n faults (if we consider SA0 and SA1 at all the nets).
The below example explains in detail about functional and structural testing.

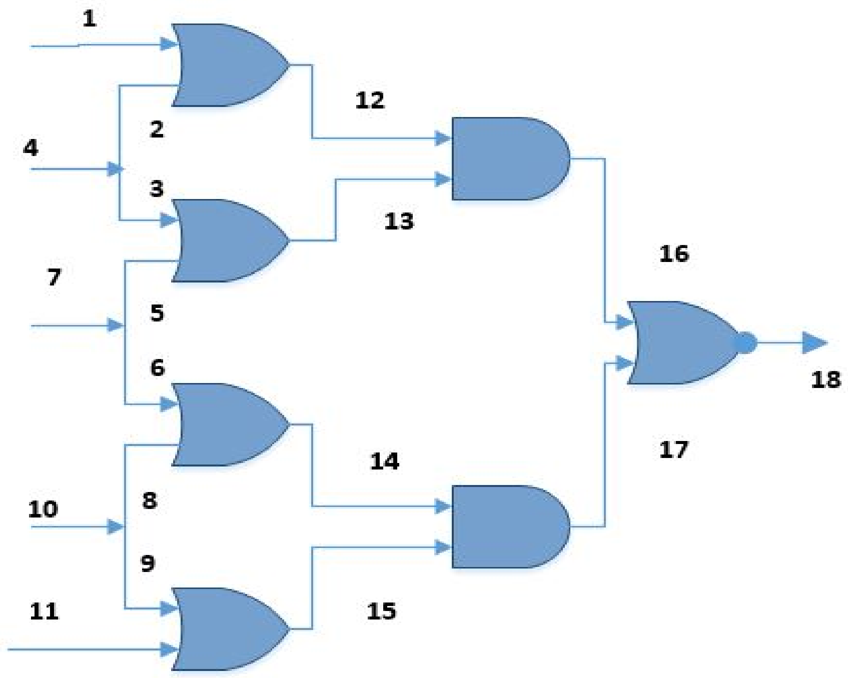


Figure 3.15: structural and functional fault testing

In every circuit there are many nets connecting every device. Each net has the possibility to have stuck-at-1 or stuck-at-0 fault. In the given circuit there are 18 net lines, so there are possibility of having 36 stuck-@ faults. For functional Testing we need to give all the combination of input. Here there are 5 inputs, so the no. of inputs required to functionally check the circuit are 2^5. In case of Structural testing we just check for the nets not for the functionality of the circuit. So the no of test vector we need to give at the input is just 36 (2 x no. of stuck-@ faults).

3.14 Summary

Fault models are important for testing methodology. Stuck-at fault models are the best model which covers all the models. Fault Equivalence and dominance are the techniques to reduce the fault list. Testing is equally important as all other manufacturing steps. Structural functional are preferred over functional testing to save time, cost and memory space.

Chapter 4

Fault Coverage Escalating Methodology

4.1 Fault Grading

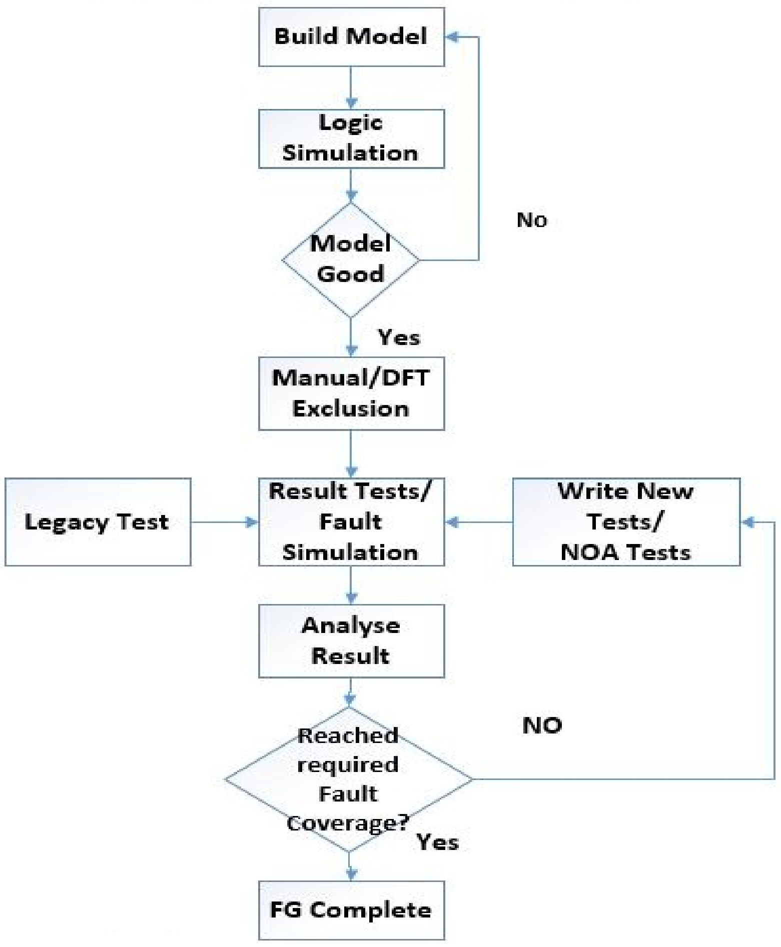


Figure 4.1: Basic Fault Grading Flow

Fault grading (FG) is the entire process of deriving a series of tests, and evaluating/ grading their effectiveness in detecting possible manufacturing defects. FG has two purposes, one to serve as a process for test development, and the other to measure and improve the quality production tests. The flow of Fault Grading is as showing below

4.2 Model Building

A new model which is a gate level model is developed for doing gate level simulations. The model conversion was performed using internal Intel computer aided design tool. The tool generates a simulation model (which is functionally equivalent to the RTL model, but with a different level of abstraction) from Intel's schematic netlist. The model developed is tuned to compensate for the limitations of the simulation. Model building involves: assigning directions to the pins of a transistor, and ignoring and adding elements.

4.3 Logic Simulation or Logic Validation

Executing the logic model with all tests in the test suite, thus verifying the model and the tests, by comparison to other simulator results is called logic simulation or logic validation. Once the Logic model simulates correctly, Fault Simulation is carried out. On this model all the legacy tests are run and the status of the faults detected is changed to DET.
Legacy tests are tests which are inherited from the previous generation of the processors. There might be compatibility issues with the legacy tests. Tests can fail for various reasons. The debugging phase involves identifying the reasons for the failure, grouping the tests failing for the same reason into separate categories and then fixing the errors.Once the logic model simulates correctly, fault simulation can begin

4.4 Exclusion

To carry out effective Fault Grading, we need to exclude some of the faults like, un-testable faults, DFT related logics and BIST logics. This help in boosting the fault coverage. The Exclusion is done by many ways:

4.4.1 Manual Exclusion

In this process, we exclude the faults which are mainly for DFT. The UnDetectable(UND) are converted to Untestable(UNT) is updated in the fault Database

4.4.2 DFT Signal Constraints

In this, the DFT signals are first extracted from the design and then constrained to a particular constant value. In an ideal case, the DFT signals will have a constant value when a functional test is run, because functional tests should not affect the DFT units. Also constraining a signal to a particular value will help in tracking the path of the signal. All the nodes where this signal is propagated will then be converted into UNTs, if they are UNDs in the fault database. The list of the signals to be constrained along with the value to which they are constrained are added onto a constraint (.const) file and then submitted for implementation.

4.4.3 BIST exclusion

In this, all the Logic chips such as RAM and microprocessor units are included with BIST(Built in self tests) in the circuit. BIST logics are inserted to detect any defective array of elements.

4.5 Execute Tests/Faults Simulation

Fault Grading uses fault simulation for grading the tests in the test suite. This can be explained using the following block diagram shown in fig 4.2
Fault grading uses a database of all the faults along with its status like detects (DET), undetectable (UND), unobservable (UNO) and untestable (UNT) etc. Whenever a fault is detected, its status will be changed to DET in the database. Faults which are already detected will not be considered for fault simulation later. Only UND faults will be simulated by the fault simulator.

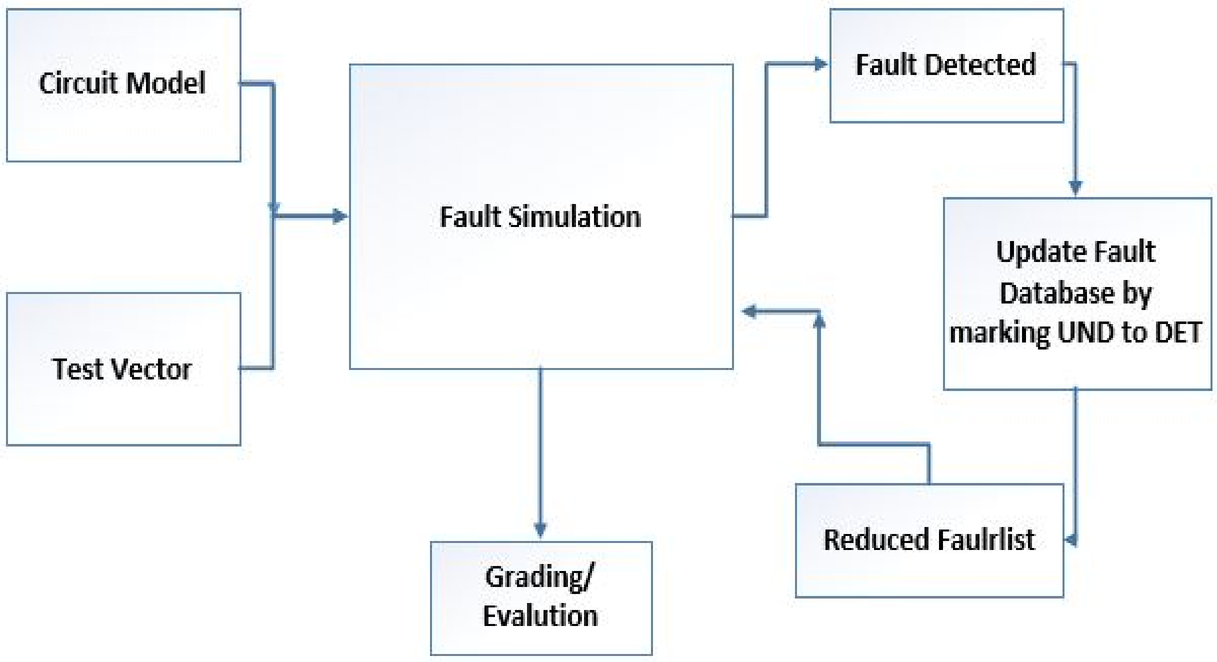


Figure 4.2: Fault Simulation Process

4.6 Grading and Evaluation

Evaluating fault simulation results and grading the effectiveness of a set of tests will be used to determine fault coverage. Fault coverage calculated in this step is checked against the required coverage. If the coverage requirements are not met, the process is rerun from enhancing the test suite or write new tests for the unit where the fault coverage is lagging.

4.7 NOA (Node Observability Architecture)

It is very difficult to test the given circuit, as the circuit is very large to check its individual nodes. If a fault is detected at the output of the circuit it is very difficult to locate the fault. For that purpose we use the method of NOA. It is one of the method to enhance the fault coverage.
We try to add some observation points in between the nodes of the circuit, where we can observe the state at a particular time. In every unit of graphics there is a NOA functional block. In that NOA functional block there is a NOA register, which takes note of different observation points in the circuit.

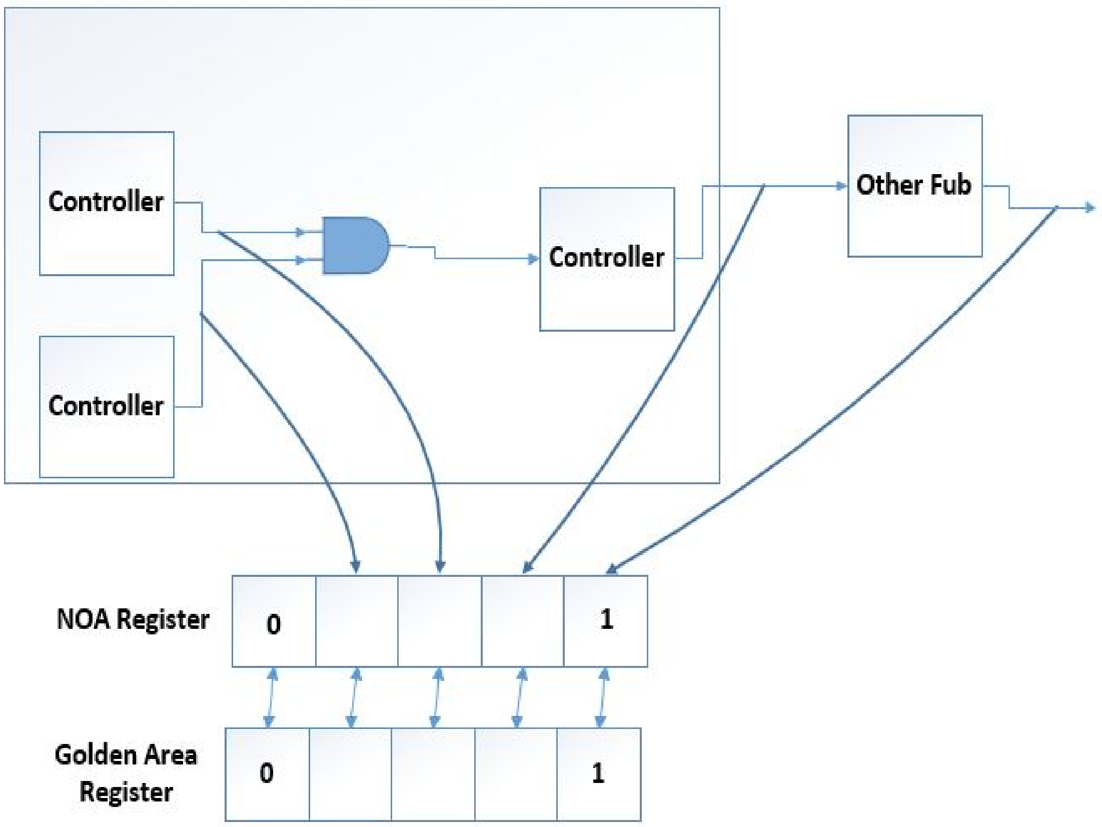


Figure 4.3: NOA register

As shown in the Fig 4.3, the output from all the controllers are stored in a NOA register, inside the NOA fub. The results of these Noa registers are then compared with Golden Area to check whether the output is same as the golden area or not.
From the list of tests for a particular functional block, top 5 to 6 tests which toggles maximum time are used to check for NOA for that particular functional block. This will help to increase the percentage of faults detect in that area.

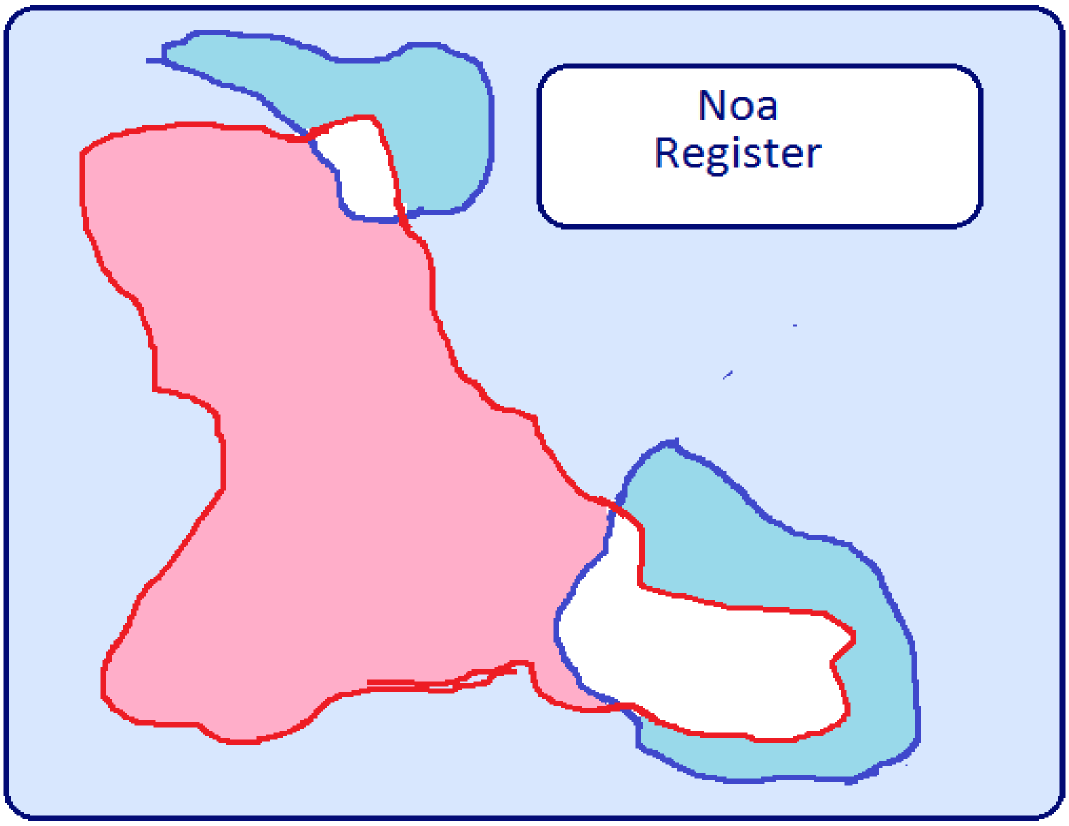


Figure 4.4: Region Covered by simple tests and NOA tests

The blue portion is the portion hit by the normal tests and the red portion is the portion hit by tests with NOA . The portion in white is the common portion hit by both the tests. By using NOA we Explore more area and try to detects more faults so that it can help in enhancing the fault coverage of the unit. Using NOA we try to detects more unique detects. For example: Suppose Normal tests have given 4 detects and NOA tests has given 10 detects which also contains the 4 detects found by the normal test then NOA tests has given 6 unique detects.

4.8 Summary

In this chapter we discussed the basic methodology for increasing the fault coverage. The initial step is to build a model then to check for its logic validity. To reduce the fault list we carry exclusions. Next step is to do Fault Simulation which is the important step for fault grading. Writing NOA tests to increase the Fault Coverage to hit the areas which are not reaching their required Fault Coverage goal.

Chapter 5

Tools and Technology

In this chapter we will look into the tools we used for fault grading, the results we got and their importance.

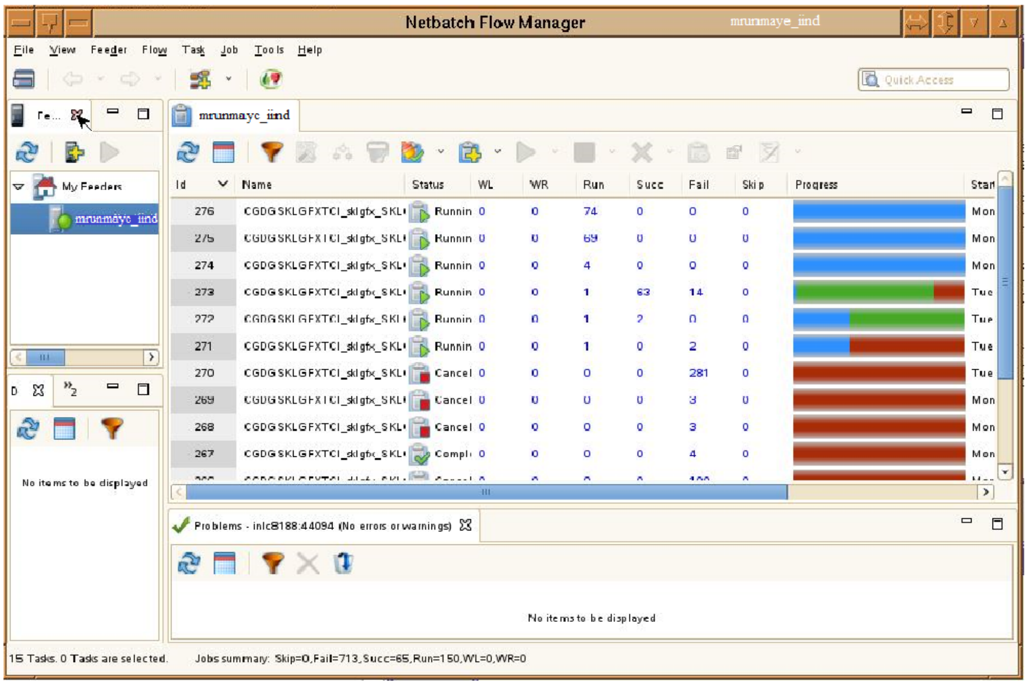
5.1 Regression

We carry out regression of the test vectors, whether new or legacy, and check for the output. In the output we get few files, they are RunLog File, report File, and fsdb file. Regressions are carried out by using run command in Unix.

5.1.1 Run command

Run command is the command used to run the tests. Tests can be run in 2 ways, depending on the number of the tests need to be run. If a set of tests we need to run then we need to launch the tests netbatch mode, wheseas single test can be launched on local machine in normal mode.

- **Netbatch Mode:**
  Netbatch mode is used to run a group of tests. The tests submitted to netbatch are sent to a particular queue. They start running as they reach at the top of the queue, depending on the available free machines in the pool.
  The netbatch manager tool is used to monitor the status of regressions. As shown in the figure it shows the pass rate and error rate.



Netbatch Flow Manager

Id	Name	Status	WL	WR	Run	Succ	Fail	Skip	Progress	Start
276	CGDGSKLGFXTCI_sklgfx_SKL	Runnin	0	0	74	0	0	0	🟦	Mon
275	CGDGSKLGFXTCI_sklgfx_SKL	Runnin	0	0	69	0	0	0	🟦	Mon
274	CGDGSKLGFXTCI_sklgfx_SKL	Runnin	0	0	4	0	0	0	🟦	Mon
273	CGDGSKLGFXTCI_sklgfx_SKL	Runnin	0	0	1	63	14	0	🟩🟥	Tue
272	CGDGSKLGFXTCI_sklgfx_SKL	Runnin	0	0	1	2	0	0	🟦🟩	Tue
271	CGDGSKLGFXTCI_sklgfx_SKL	Runnin	0	0	1	0	2	0	🟦🟥	Tue
270	CGDGSKLGFXTCI_sklgfx_SKL	Cancel	0	0	0	0	281	0	🟥	Tue
269	CGDGSKLGFXTCI_sklgfx_SKL	Cancel	0	0	0	0	3	0	🟥	Mon
268	CGDGSKLGFXTCI_sklgfx_SKL	Cancel	0	0	0	0	3	0	🟥	Mon
267	CGDGSKLGFXTCI_sklgfx_SKL	Compl	0	0	0	0	4	0	🟥	Mon

Problems - inlc8188:44094 (No errors or warnings)

No items to be displayed

15 Tasks. 0 Tasks are selected. Jobs summary: Skip=0,Fail=713,Succ=65,Run=150,WL=0,WR=0

Figure 5.1: Regression Status in NetBatch Flow Manager

- **Local Mode:**
  Local interactive mode is used for running single test interactively without waiting for long in netbatch queue.

5.2 Test Debugging

Test debugging is difficult and phase in fault grading flow. If tests fails the n we need to debug the tests by getting to the root cause of the error. Some of the errors faced are mentioned below:

5.2.1 Checker Error

The first step in debugging the fails is to check the report files and the log files and find out whether there are any checker errors. If there are any checker errors then we need to open the golden reference dump and corresponding RTL tracker file and check for the mismatch. Also we need to open the full signal dump file to track the signal value. This is how we will be to track the checker error.

5.2.2 Run Limit Error

If the test fails with the error signature run limit reached, we need to check whether it is a test hang or whether it requires more simulation time. We can check this by viewing the most recent tracker file transactions and check whether the time stamp of the last transaction is near to the assigned run limit. If the time stamp of the last transaction is not near to the run limit then it is a test hang. For example if the run limit assigned for a test is 5ms and the time stamp of the last transaction is 4.99ms, it means that the test needs more time to simulate. So increasing the run time limit will be the option to overcome this error.

5.2.3 Test Programming Error

This error occurs when there is any programming syntax error or there is any error in writing the required format for the given version. Intel uses its proprietary language, called Graphics Random Instruction Tests. It uses Ruby environment.

5.3 Test_Status

This tool is used to check the tests and FUB status as well as test is submitted to database using

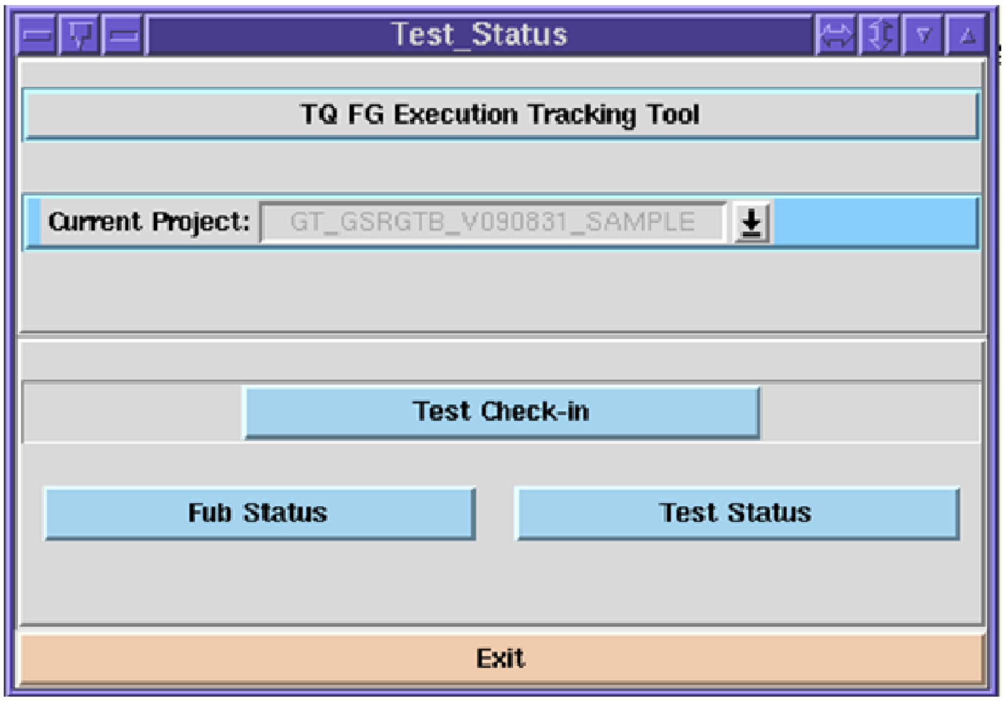


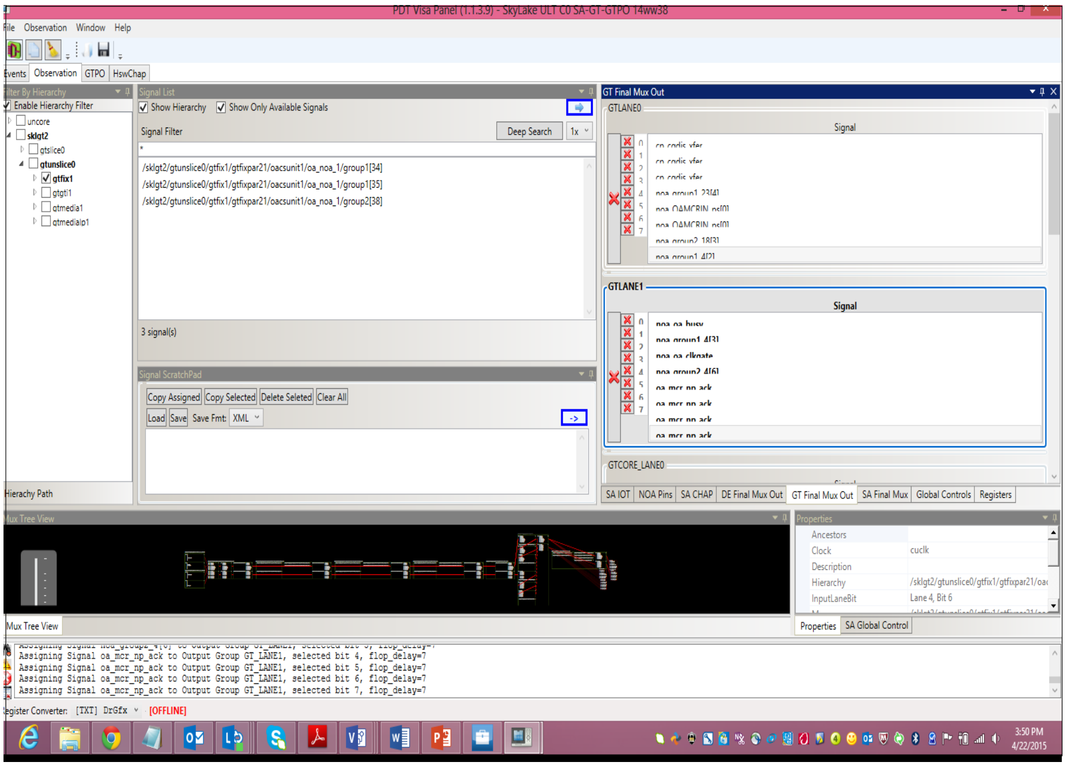
Figure 5.2: Regression Status in NetBatch Flow Manager

The different sub-categories of the tool are as follows:

- Test Check in: IT is used to check in the tests into the data base which have detected some faults and have incremented the fault coverage %.
- FUB Status: This sub-category helps in detecting the Fault coverage on a per module basis, in terms of number of DET, UNT UNO faults and net fault coverage of the given FUB or module.
- Test status: In Test Status, search for the expression related to test case, in terms of QID or the unit on which the test case is targeting.

5.4 Fixing NOA

Using Platform Debugging Tool, which is Intel's proprietary tool, we can cover some of the faults on design by adding some of the observation points in the RTL design, that will help us to know the status at some particular nodes.



```
*
/sklgt2/gtunslice0/gtfix1/gtfixpar21/oacsunit1/oa_noa_1/group1[34]
/sklgt2/gtunslice0/gtfix1/gtfixpar21/oacsunit1/oa_noa_1/group1[35]
/sklgt2/gtunslice0/gtfix1/gtfixpar21/oacsunit1/oa_noa_1/group2[38]

3 signal(s)
```

Ancestors	
Clock	cuclk
Description	
Hierarchy	/sklgt2/gtunslice0/gtfix1/gtfixpar21/oa
InputLaneBit	Lane 4, Bit 6

```
Assigning Signal oa_mcr_np_ack to Output Group GT_LANE1, selected bit 4, flop_delay=7
Assigning Signal oa_mcr_np_ack to Output Group GT_LANE1, selected bit 5, flop_delay=7
Assigning Signal oa_mcr_np_ack to Output Group GT_LANE1, selected bit 6, flop_delay=7
Assigning Signal oa_mcr_np_ack to Output Group GT_LANE1, selected bit 7, flop_delay=7
```

Figure 5.3: Regression Status in NetBatch Flow Manager

Some of the steps on how to add observations points:

1. First step is to identify which Units are lagging coverage by using Fault coverage analysis.

2. Get the list of all the undetectable faults in the unit.
3. Trace forward and backward paths for each of the faults.
4. Find out the maximum toggling signal, and to which flip-flop it goes to.
5. Check whether that flip-flop is connected to NOA register. (Observation point)
6. If not then connect it to the NOA register

5.5 Summary

This chapter covers all the tools used in this project and some of the common errors faced during launching the tests. It also covers all the stepwise instruction that is needed to be followed while using the tools. Each tool used has some specific use as explained in the chapter.

Chapter 6

Results and Analysis

6.1 Fault Coverage

These are some of the results obtained for different units of Intel's SkyLake architecture Graphics Processor at different tests levels.

6.1.1 Sampler Cluster

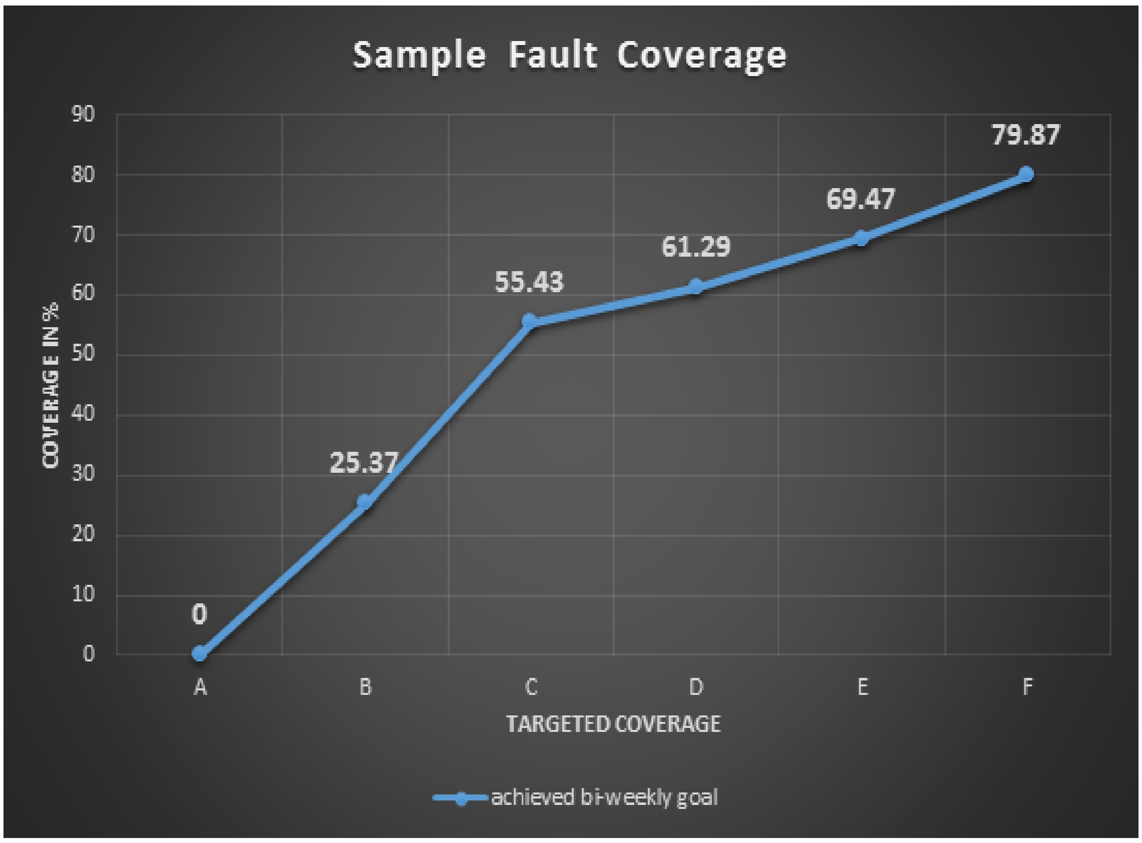


Figure 6.1: Sampler Cluster Fault Coverage Chart

A : At the start
B : Basic Tests
C : Legacy Tests
D : Modified Tests
E : Excluding DFT Logics
F : NOA Tests

Above Chart shows the bi-weekly fault coverage goal of the Sampler cluster. The goal was achieved by writing good tests and debugging the legacy tests.
Debugging includes checking for the syntax, missing important files, Loader fitting problems, etc.

6.1.2 Pipeline Cluster

The below is the chart for the bi-weekly goal for pipeline cluster.

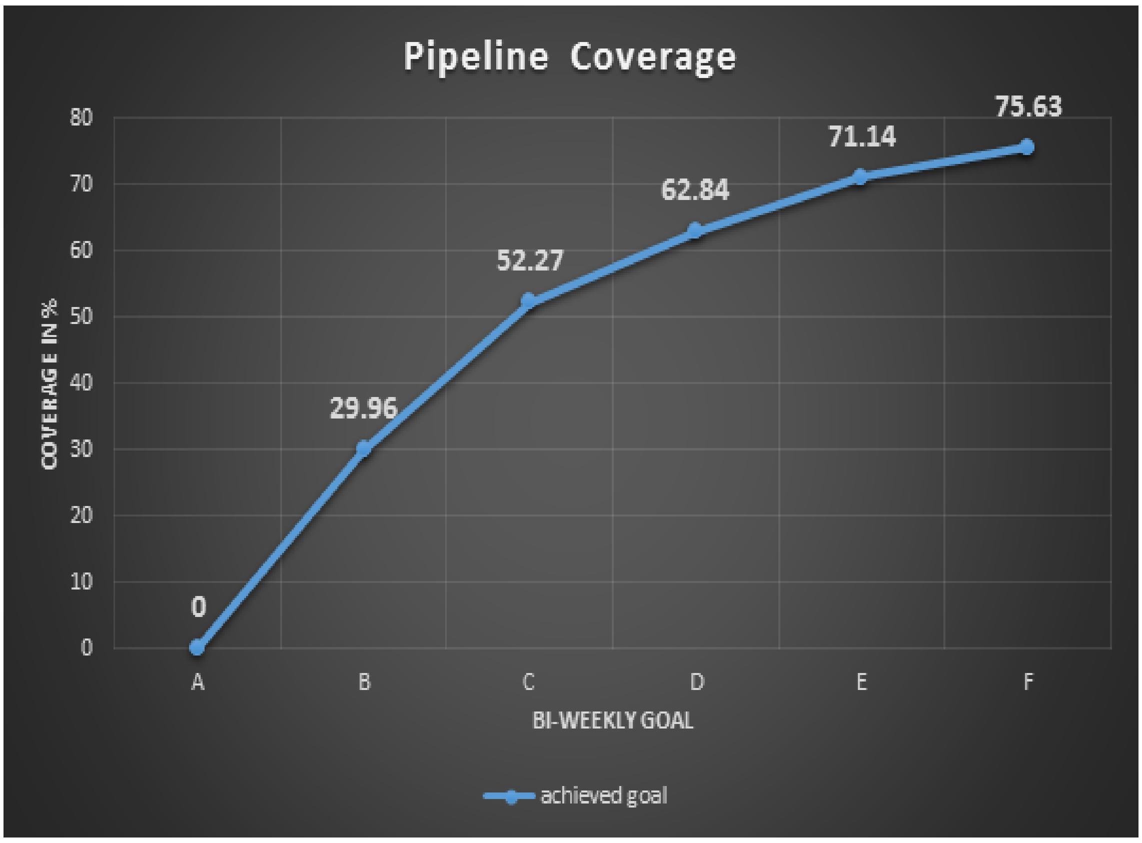


Figure 6.2: Pipeline Cluster Fault Coverage Chart

A : At the start
B : Basic Tests
C : Legacy Tests

D : Modified Tests
E : Excluding DFT Logics
F : NOA Tests

6.1.3 Interface Unit

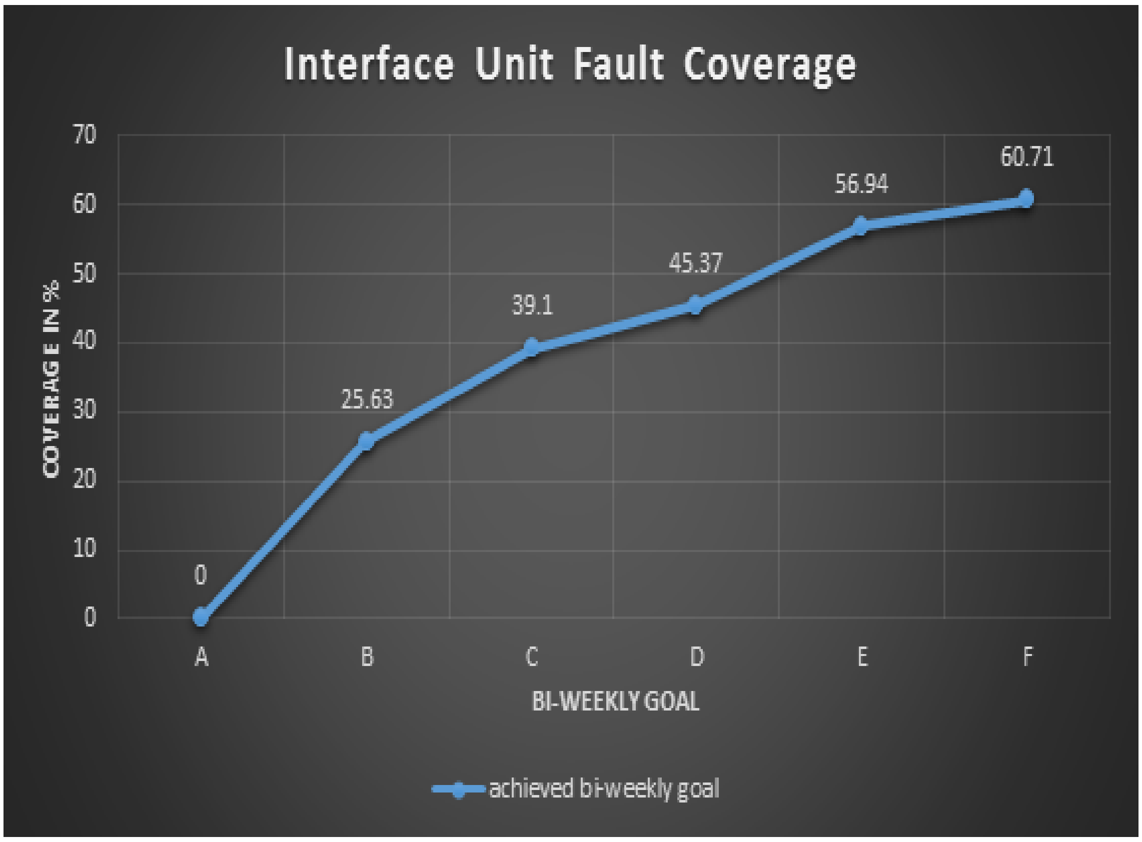


Figure 6.3: Interface Unit Fault Coverage Chart

A : At the start
B : Basic Tests
C : Legacy Tests
D : Modified Tests
E : Excluding DFT Logics
F : NOA Tests

The numbers are not exact as per Intel's Confidencial policy.

Chapter 7

Conclusion and Future goal

7.1 Conclusion

This project work came up with a methodology to develop test content to meet the fault coverage target. It also explained the utility of the fault grading to find out the areas in the design which lack coverage and develop test cases for effective fault detection. The tests that are developed during the fault grading process are converted to traces which are in turn tested out on a tester. Graphics Processing Unit is an important component in multimedia technology. The internal structure of the same were studied in detail. It makes achieving the required coverage easier. Fault Grading is an important process to detect faults. Fault Simulation helps in developing high quality manufacturing tests and thus reducing the number of defective parts shipped to the customer. Thus, this methodology can be used to increase the quality of the manufactured products in terms of DPM (defects per million).

7.2 Future Goal

In some Logics, analysis was done regarding the inability faced to detect faults. This shows lack of observation points in these logics. Thus the future scope will be to insert some more observation points in the design so that maximum faults are detected and required fault coverage is reached.

APPENDIX

1. Legacy Tests: Tests which are inherited from the previous generation of processors.
2. DFT: DFT stands for Design for Testability. This is a name for design techniques that add certain testability features to a microelectronic design.
3. Signal: A signal refers to the connections between components in a logic model, like wires.
4. Test Vector: A test vector is a single test input or the collection of all the input values at a given time.
5. Test: Test is a sequence of input vectors which when simulated, produces a sequence of output vectors.
6. Test Suite: A test suite is an ordered series of tests.
7. Good Machine: Good machine is a defect free copy of the circuit used as the basis for comparison during fault simulation
8. Faulty Machine: Faulty Machine represents the behavior of the circuit on which a fault has been inserted

Bibliography

[1] Intel Documents

[2] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing, ”VLSI Test Principles and Architecture Design for Testability ”, CRC Press, Second Edition, 2007

[3] Weiwei Mao, and Ravi K. Gulati, â€œImproving Gate Level Fault Coverage by RTL Fault Gradingâ€, International Test Conference, 1996, Paper 6.3, pp 150-159

[4] Kevin Kriwell, ”Difference between CPU and GPU” , http://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/ , December 16, 2009

[5] Michael L Bushnell and Vishwani D Agarwal “ Essentials of Electronic Testing”, Kluwer Academic Publishers, 2002

[6] Yu Zhang, ”Diagnostic Test Pattern Generation and Fault Simulation for Stuck-at and Transition Faults” , http://www.eng.auburn.edu/ãgrawvd/THESIS/ZHANG/dissertation_yu_2012.pdf, August 4, 2012

[7] Mann Bhammar, ” Functional Testing and structural Testing” , http://neurontesting.blogspot.com/2012/05/functional-black-box-testing-and.html May 1, 2012

[8] Ashu Rege, ”Modern GPU Architecture” , ftp://download.nvidia.com/developer/cuda/seminar/TDCI_Arch.pdf , 2008 imp http://haifux.org/lectures/267/Introduction-to-GPUs.pdf

[9] Naga Gollakota and Ahmed Zaidi Fault Grading of Intel 80486, 1990 International Test Conference, Paper33.3, pp 758-761

[10] Prof.Roger Crawfis, https://www.google.com/search?hl=enq=%22transistor%20counts,%20while%20CPUs%20have %20few%20execution%20units%20and%22cd_min=1%2F1%2F2000cd_max=1%2F1%2F2015source =lnttbs=cdr%3A1gws_rd=ssl , 27 Feb, 2007

[11] Michel Vallières, http://physics.drexel.edu/w̃king/courses/phys405_f10/content/GPUs/ ,19 March, 2012

[12] Ricardo Veguilla, 8 May 2010, https://www.google.com/search?hl=enq=%22is%20the%20science, %20study,%20and%20method%20of%20projecting%20a%20mathematical%22cd_min=1%2F1%2F2 000cd_max=1%2F1%2F2015source=lnttbs=cdr%3A1gws_rd=ssl

[13] Intel Tests Methodology Handbook, Pg 85-185, 28 March, 2008

[14] http://www.osu-tulsa.okstate.edu/istr/FaultModelling/chapter2JMAdissertation.pdf