# Fault Coverage Improvement for Advanced Graphics Processor

**Major Project Report**

Submitted in partial fulfillment of the requirements

for the degree of

**Master of Technology**

**in**

**Electronics & Communication Engineering**

**(Communication Engineering)**

By


**Shaishavkumar Gandhi**

**(13MECC05)**

**Electronics & Communication Engineering Branch**

**Electrical Engineering Department**

**Institute of Technology**

**Nirma University**

**Ahmedabad-382481**


**May 2015**

# Fault Coverage Improvement for Advanced Graphics Processor

## Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

**Master of Technology**

**in**

**Electronics & Communication Engineering**

**(Communication Engineering)**

By


**Shaishavkumar Gandhi**

**(13MECC05)**


Under the guidance of


**External Project Guide:**
**Mrs. Kavitha Seshadri**
Manager,
Intel Technology India Pvt. Ltd,
Bangalore.

**Internal Project Guide:**
**Dr. Yogesh N. Trivedi**
Professor(EC Dept.),
Institute of Technology,
Nirma University, Ahmedabad.

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

# Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Communication Engineering 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.

**- Shaishavkumar Gandhi**
**(13MECC05)**

# Disclaimer

"The content of this report does not represent the technology, opinions, beliefs or positions of Intel Company, its employees, vendors, customers or associates."

# Certificate

This is to certify that the Major Project entitled **"Fault Coverage Improvement for Advanced Graphics Processor"** submitted by **Shaishavkumar Gandhi (13MECC05)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Communication Engineering, Nirma University, Ahmedabad is the record of work carried out by him 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.

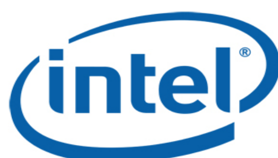
Date: Place: Ahmedabad

**Dr. Yogesh N.Trivedi**
Guide

**Dr. D.K.Kothari**
Program Coordinator

**Dr. P.N.Tekwani**
Head of EE Dept.

**Dr. K.Kotecha**
Director, IT



## Certificate

This is to certify that the Major Project "**Fault Coverage Improvement for Advanced Graphics Processor**" submitted by **Shaishavkumar Gandhi (13MECC05)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Communication Engineering, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Date:

Place: Bangalore

Mrs. Kavitha Seshadri,
Manager,
Intel Technology India Pvt.Ltd.
Bangalore.

# Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. P.N.Tekwani**, Head of Electrical Engineering Department,**Dr. D.K.Kothari**, Coordinator of M.Tech Comminication Engineering program and **Dr. Yogesh N.Trivedi** for allowing me to undertake this thesis work and for his guidelines during the review process.

I am deeply indebted to my thesis supervisors, **Mrs. Kavitha Seshadri**, Manager,Intel Technology India Pvt. Ltd. and **Dr. Yogesh N. Trivedi**, Professor,EC Department, Nirma University for their constant guidance and motivation. I also wish to thank all other team members at Intel for their constant help and support. Without their experience and insights, it would have been very difficult to do quality work.

I would like to extend sincere thanks to my mentor **Mr. Madhu B.** from Intel for guiding me throughout the project and providing valuable suggestions. He has been a constant source of knowledge and motivation to me.

I wish to thank my friends of my class for their delightful company which kept me in good humor throughout the year.

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.

**- Shaishavkumar Gandhi**
**13MECC05**

# Abstract

The VLSI industry is growing according to Moore's law. As a result, an integrated circuit designs are going towards more complexity due to which testing becomes tedious and tough. A VLSI chip may fail due to many reasons like design or fabrication defects, environmental factors or a combination of these. The physical defects may consist of break in lines, short between lines at the interconnection level, shorts to substrate, point defects and imperfections such as scratches across the chip. In order to detect the faults that are induced during manufacturing, good quality tests are required. So devising good tests is one of the most important steps in manufacturing quality micro-circuits. The quality of the test is represented by its fault coverage through the fault simulation process. The effective way to test a circuit is to observe its behavior by building a logical model and then using these logical models to check the physical characteristics of the circuit. These logical models are called as the fault models.

Fault Coverage is a quantitative measure of test grading. Fault Grading is the entire process of deriving a series of tests, and evaluating/grading their effectiveness in detecting possible manufacturing defects.

This project work aims to come up with a methodology to meet the desire fault coverage target of 81% for the sampler and execution unit of Intels Graphics Processor Unit (GPU) by using various fault analysis techniques.

# Contents

**Declaration** **iii**

**Disclaimer** **iv**

**Certificate** **v**

**Acknowledgements** **vii**

**Abstract** **viii**

**List of Figures** **xiii**

**1 Introduction** **1**

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Area of the work . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.3 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.4 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Literature Survey** **4**

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 Graphics Fundamentals . . . . . . . . . . . . . . . . . . . . . . . 4

2.2.1 Graphics Processing Unit (GPU) . . . . . . . . . . . . . . . . 6

2.2.2 GT Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.3 3D Graphics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.4 3D Graphics Pipeline . . . . . . 11
2.4.1 Vertex Fetch . . . . . . 11
2.4.2 Vertex Shader . . . . . . 11
2.4.3 Hull Shader . . . . . . 13
2.4.4 Tessellator . . . . . . 13
2.4.5 Domain Shader . . . . . . 13
2.4.6 Geometry Shader . . . . . . 14
2.4.7 Resterizer . . . . . . 14
2.4.8 Pixel Shader . . . . . . 15
2.4.9 Pixel Output . . . . . . 15
2.5 High volume manufacturing and test related Concepts . . . . . . 15
2.5.1 Introduction . . . . . . 15
2.5.2 High Volume Manufacturing . . . . . . 15
2.6 Structural vs. Functional Testing . . . . . . 18
2.7 Fault Model . . . . . . 18
2.8 Circuit Defects and Fault Modeling . . . . . . 19
2.9 Types of Models . . . . . . 19
2.9.1 Stuck at model . . . . . . 20
2.9.2 Types of S@faults . . . . . . 20
2.10 Fault Collapsing . . . . . . 21
2.11 Fault Classification . . . . . . 21
2.12 Fault Coverage . . . . . . 22
2.13 Concept of Fault Simulation . . . . . . 22
2.14 Fault Simulation Algorithm . . . . . . 23
2.14.1 Serial Fault Simulation Algorithm . . . . . . 23
2.14.2 Parallel Fault Simulation Algorithm . . . . . . 25
2.14.3 Concurrent Fault Simulation Algorithm . . . . . . 26
2.15 Fault Sampling . . . . . . 27
2.16 Fault Detection . . . . . . 28

2.17 Summary . . . . . . . . . . 30

**3 Fault Grading** **31**

3.1 Introduction . . . . . . . . . . 31

3.2 Fault Grading Methodology . . . . . . . . . . 31

3.2.1 Build Fault Model . . . . . . . . . . 33

3.2.2 Running all Legacy tests on this model . . . . . . . . . . 33

3.2.3 Find good tests from legacy tests . . . . . . . . . . 33

3.2.4 Implement Exclusions . . . . . . . . . . 33

3.2.5 Write new tests . . . . . . . . . . 34

3.3 Randomized test generation . . . . . . . . . . 34

3.4 Test Execution Flow . . . . . . . . . . 34

3.5 Test Debug . . . . . . . . . . 35

3.5.1 Checker error . . . . . . . . . . 36

3.5.2 Run limit reached . . . . . . . . . . 36

3.5.3 L3-Loader Fit-ability Issues . . . . . . . . . . 36

3.6 Node Observability Architecture(NOA) . . . . . . . . . . 36

3.7 Summary . . . . . . . . . . 39

**4 Results and Analysis** **40**

4.0.1 Regression Results . . . . . . . . . . 40

4.0.2 Run Command . . . . . . . . . . 41

4.0.3 Fault Coverage Result of Sampler unit . . . . . . . . . . 42

4.1 Summary . . . . . . . . . . 43

**5 Conclusion and Future Scope** **44**

5.1 Conclusion . . . . . . . . . . 44

5.2 Future Scope . . . . . . . . . . 45

**References** **46**

**A Appendix** **47**

# List of Figures


2.1 The effects of better resolution [1] . . . . . . . . . . . . . . . . . . . . . . 5
2.2 2D image and 3D image of an elephant [1] . . . . . . . . . . . . . . . . 6
2.3 Block Diagram of Graphics Controller [1] . . . . . . . . . . . . . . . . . 7
2.4 Graphics technology model . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.5 3D Pipeline Block Diagram . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.6 High Volume Manufacturing Flow[1] . . . . . . . . . . . . . . . . . . . . 16
2.7 Fault Simulation Process . . . . . . . . . . . . . . . . . . . . . . . . . . 23
2.8 Serial Fault Simulation Algorithm . . . . . . . . . . . . . . . . . . . . . 24
2.9 Parallel Fault Simulation Algorithm . . . . . . . . . . . . . . . . . . . . 25
2.10 Concurrent Fault Simulation Algorithm . . . . . . . . . . . . . . . . . . 27
2.11 Example showing how a test vector cannot detect a fault . . . . . . . 29
2.12 Example showing how a test vector detects a fault . . . . . . . . . . . 29

3.1 Flow chart of fault grading[7] . . . . . . . . . . . . . . . . . . . . . . . . 32
3.2 Flow of test execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
3.3 Working principle of NOA . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.4 Coverad area with NOA Tests . . . . . . . . . . . . . . . . . . . . . . . 38

4.1 Validation Flow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
4.2 Netbatch flow manager GUI . . . . . . . . . . . . . . . . . . . . . . . . 41
4.3 Graph showing Fault Coverage progress. . . . . . . . . . . . . . . . . . 43

# Chapter 1

# Introduction

## 1.1 Motivation

High Volume Manufacturing is the most commonly used manufacturing technique for Integrated Circuits. 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 micro-circuits. 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 for advancement in 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 (DPM) 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 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 Area of the work

In todays fast growing as well as increasing complex world of VLSI circuits, test quality has significant effect on the quality of the digital circuit. High quality tests can gives batter screening and discovery of defective chips before they leave the manufacturing plant. High quality testing minimizes DPM and thus can significantly minimizes manufacturing costs and increase company reliability.Thus, our main area of work in this project is to find bugs or faults from a digital circuit and minimize those bugs or faults to an acceptable level.

## 1.3 Problem Statement

The main objective of this project work is to come up with a required fault coverage target using various fault grading methodologies. Also to develop the test content that will meet the desired fault coverage target. This test content will be facilitating the post silicon validation process to test the manufactured chips for defects. We wish to meet the fault coverage target of 81% for the sampler and execution unit of Intels Graphics Processing Unit (GPU) by using various fault analysis techniques.

The aim of this project is to

1. Study about internal structure of graphics processor to improve test content.

2. Running regressions of those tests to validate different partitions of the GPU.

3. Report bugs from faied tests and debug them with appropriate fix.

4. By the help of Fault Grading result, try to improve overall fault coverage of the GPU.

## 1.4 Thesis Organization

The thesis report is organized as follow:

1. The second chapter discusses about literature stydy in such areas like the graphics fundamentals ,3D Graphics model with its fundamentals, the 3D graphics pipeline.It also gives an idea about the High Volume Manufacturing (HVM) concepts, the test related concepts such as fault modeling, types of fault models, fault classification, fault coverage, concept of fault simulation and their significance

2. The third chapter explains the Fault Grading (FG) methodology. It also explains the methodology followed for the fault grading of sampler and execution unit of Intels graphics processor.

3. The forth chapter gives information about results obtained from this project work and analysis of the same.

4. The fifth chapter summarizes the contributions of this research and provides directions for future work.

# Chapter 2

# Literature Survey

## 2.1 Introduction

In this chapter, we will discuss the literature survey done on graphics fundamentals, principles of 3D graphics and the functionality of 3D graphics pipeline.

## 2.2 Graphics Fundamentals

The building block of every computer image is called a picture element or pixel. A pixel is made up of three color components red, green, and blue. 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 memory called local memory. This type of RAM is used to store video information as it connects to both the CPU and the graphics controller.

The information in the local memory is pushed to the graphics controller when it is needed. This information is taken by the graphics controller and converted 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, as was stated earlier, 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 we can create almost any color, depending on the size of the smallest variation[1].

For example, true color (named so because the change between any two colors is smooth), can be represented by 24-bit color (16.7 million colors). Often these 24 bits are distributed evenly between each color component. With eight-bits, each color component can have 256 intensity variations! Slightly less accurate than true-color is high-color or 16-bit color which allows for 65,556 possible colors. 32-bit color is also fairly common and can yield billions of colors!

The phosphors that radiate these colors cannot, unfortunately, glow forever, so they must be continually refreshed or repainted. The electron guns must constantly scan the screen. If this scanning is not done quickly enough the screen may appear to flicker. To decrease this flickering effect, the refresh rate must be increased.

It is also quit difficult to refresh screens of higher resolution. Resolution is a measure of a screen that takes into account the number of colors and the number of pixels a monitor can display, as well as the size of the screen. A fifteen inch 24-bit screen that is 1280 pixels by 1024 pixels will be able to show much more detail than a fifteen inch 24-bit screen that can only show 640 pixels by 480 pixels. An example of this can be seen in Figure 2.1.

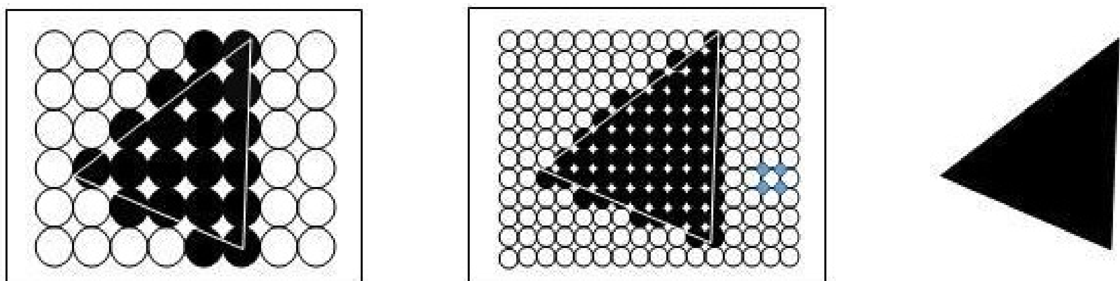


Figure 2.1: The effects of better resolution [1]

The figure 2.2 is a very simple example of the difference between 2-D and 3-D images. The 3-D image shows much more realism and depth. Some of the details of real life that add realism to 3-D images include 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 is stored which allows one to view it accurately from any angle. In the case of a 2-D picture, the viewer is limited to a single point of view.

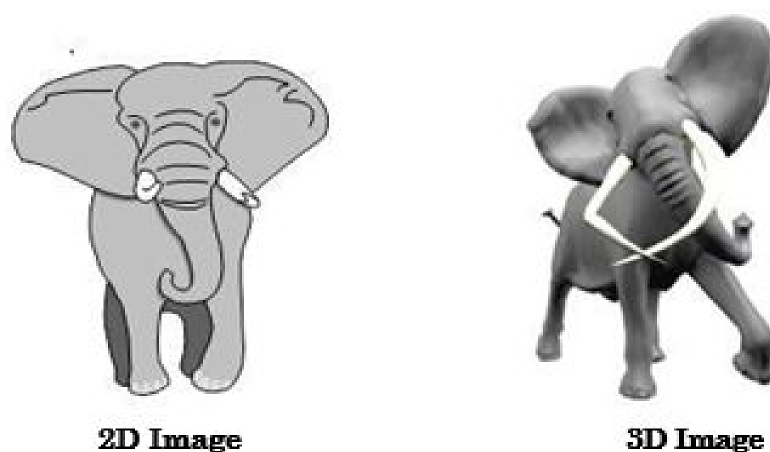



Figure 2.2: 2D image and 3D image of an elephant [1]

### 2.2.1 Graphics Processing Unit (GPU)

A GPU (also occasionally known as visual processing unit or VPU) is a dedicated microprocessor that handles and processes graphics rendering from the central microprocessor. It is used in most of embedded systems, personal computers, mobile phones, servers and game consoles etc. Modern GPUs are very effective at manipulating computer graphics and their highly paralleled structure makes them highely effective and efficient compared to general-purpose CPUs in terms of processing algorithms where large blocks of data is processed in parallel. In a personal computer, a

GPU can be present in the form of video card, or it can be onboard or in form of Soc in certain CPUs, on the CPU die. More than 90% of latest desktop and notebook computers have integrated GPUs, which are usually far less efficient and powerful compared to a dedicated video card.[1] Graphics controller is made up of 2D Engine, 3D Engine, Video Processing Engine and display interface as shown in figure 2.3. Each engine works in the different ways and based on the application, these engines will be operated to provide the better graphics and video experience.

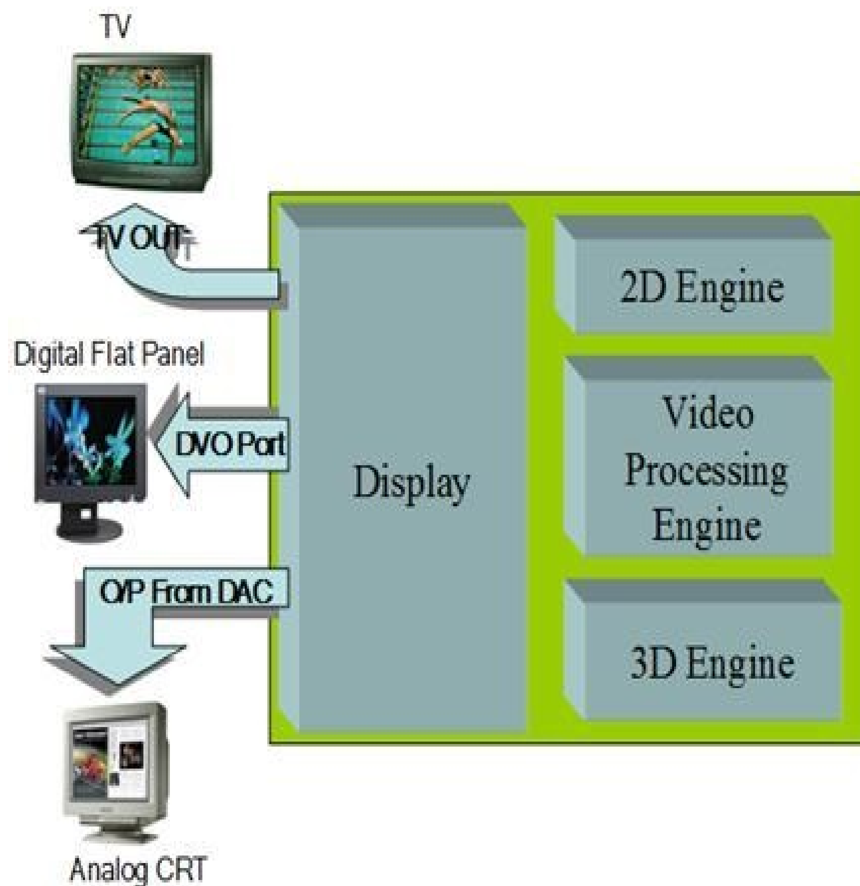



Figure 2.3: Block Diagram of Graphics Controller [1]

The subsystem having an array of execution units (EUs, referred as an array of cores) , shared functions outside the EUs that the EUs processes for I/O and for complex

computations. Programmers have access through the subsystem via the 3D or Media pipelines.
EUs are nothing but the general-purpose programmable cores which can support a rich instruction set that has been optimized to support various 3D API shader languages and media functions (primarily video) processing.
Shared functions are different hardware units.Those are used to provide specialized supplemental functionality for the EUs. A shared function is implemented on demand basis ,where the demand for a given specialized function is not sufficient to justify the costs on a per-EU basis. The specialized function is developed as a stand-alone entity outside the EUs and shared among the EUs .Invocation of the shared functionality is performed through a communication mechanism called a message. A message that is small self-contained packet of information created by a kernel and directed to a specific shared function. For SNB, the message is defined by a range of MRF registers that hold message operands, a destination shared function ID, a function-specific encoding of the desired operation, and a destination GRF register to which any write back response is to be directed. Messages are dispatched to the shared function under software control via the send instruction. This instruction identifies the contents of the message and the GRF register locations to direct any response.
The message construction and delivery mechanisms are general in their definition and capable of supporting a wide variety of shared functions.

**3D Engine:**

The 3D engines performs the following operations

1. Texture Mapping
2. Depth-Buffering (Z)
3. Fog effects
4. Lighting
5. Direct3D support

**Video Processing engine:**

It performs the following operations

1. Implements motion compensation
2. MPEG decoder
3. DXVA Support (DirectX Video Acceleration is Microsoft API specification that allows video decoding to be hardware accelerated)

**2D Engine:**

1. Handles all the 2D drawing operations.
2. Does fast data transfers called Blits to video memory.
3. GDI support. (Graphics Device Interface)

### 2.2.2 GT Model

The GT model simplifies the understanding of a GPU which is shown in figure 2.4. There are 8 clusters in all which are listed as: Execution Unit (EU), Fixed Function (FF), Slice Common (SC), L3 Cache, Media, Interface (GTI), Sampler and Half slice Data Cluster (HDC). Broadly, the model in divided into two: sliced and unsliced. The sliced part includes EU, Sampler and HDC. The slice common provides communication among the modules. Remaining all are grouped as unsliced.

## 2.3 3D Graphics

3D computer graphics uses the three-dimensional representation of geometric data stored in computer for the purposes of performing calculations and rendering 2D images. Such images will be stored back to computer to view them later or displayed

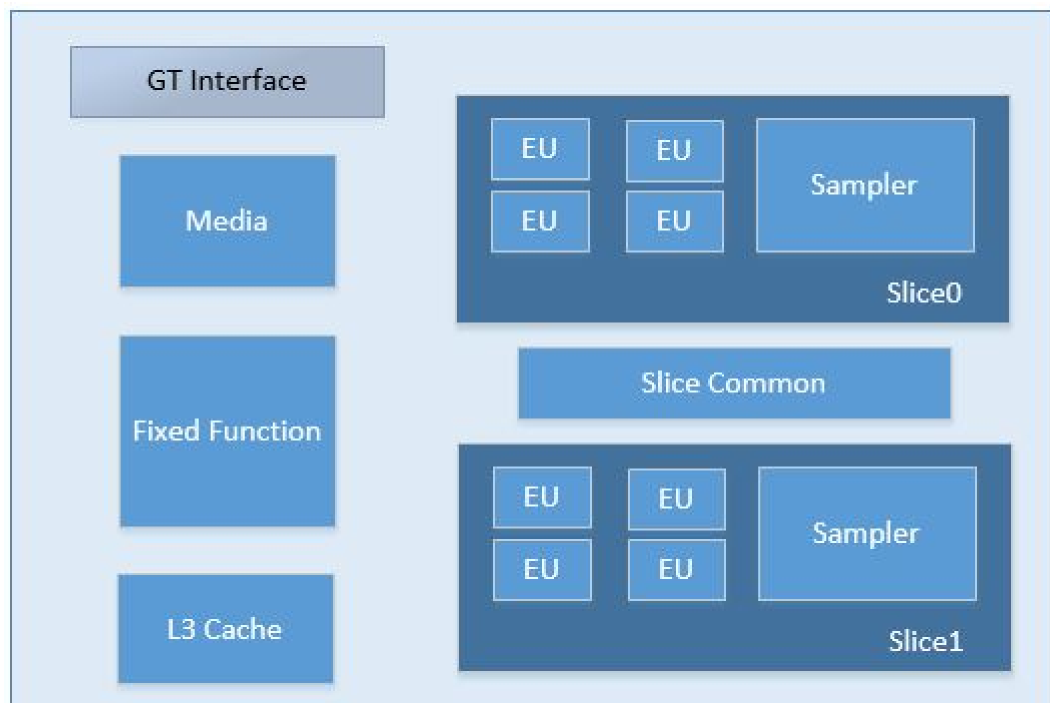



Figure 2.4: Graphics technology model

in real-time.
3D graphics is based on creating illusion of the object in real environment by using basic techniques like creating a complex object from a simple one; drawing objects only that the eye can see; using color and patterning to make them look realistic.

1. 3D graphics creates the illusion of solid objects in a real environment

2. It uses these techniques to create the illusion:

    (1) Build up complex shapes from simple ones

    (2) Only draw the objects the eye can see

    (3) Color and pattern the objects to make them look realistic

## 2.4 3D Graphics Pipeline

The figure 2.5 shows the Direct3D X rendering pipeline.In 3D computer graphics, the graphics pipeline ( or rendering pipeline) is nothing but the sequence of steps used to create a 2D raster representation of a 3D scene[2]. Once a 3D model has been created, for instance in a video game or any other 3D computer animation, the graphics pipeline is the process of turning that 3D model into what the computer displays. In the early history of 3D computer graphics, fixed purpose hardware was used to speed up the steps of the pipeline through a fixed-function pipeline. Now a days becoming more general purpose, allowing greater flexibility in graphics rendering as well as more generalized hardware.

### 2.4.1 Vertex Fetch

From the input scene, a set of primitives are generated by the application software, each of which has vertices. VS (vertex shader) transforms input/pre-shaded attributes to output/post shaded attributes. It does not change the number of vertices. It operates only on one vertex at a given time, it keeps 1:1 mapping of vertex Each vertex will have some attributes associated with it. These attributes are placed in the vertex buffers in memory by the application software and are fetched into the graphics device by the vertex fetcher unit.

### 2.4.2 Vertex Shader

For each vertex input, Vertex Shader unit generates one vertex output after applying certain arbitrary operations on the vertex attributes. Ex. Skinning, lighting etc. are the commonly used transformation operation

Major functions of Vertex shader are:

1. Transformation

2. Lightings

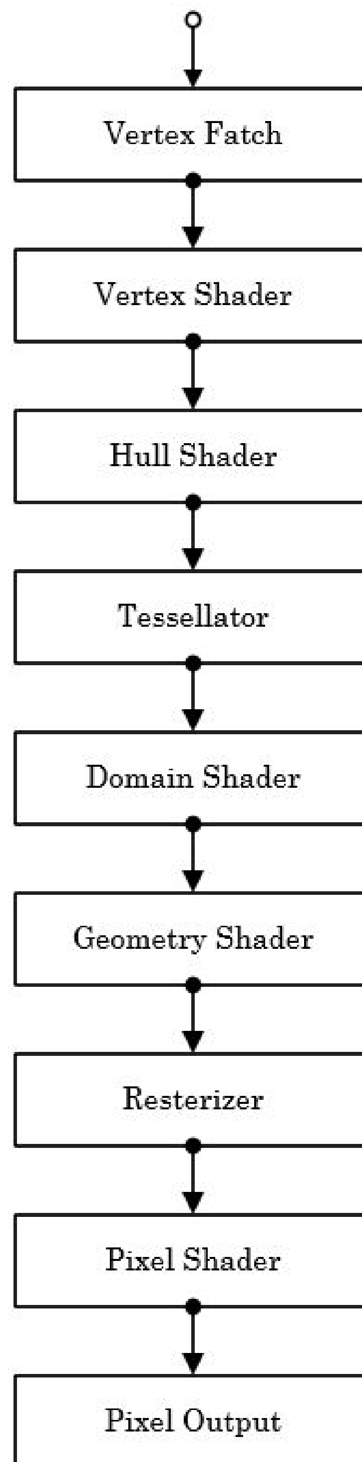



Figure 2.5: 3D Pipeline Block Diagram

a Ambient

b Diffusion

c Specular

3. Skinning

### 2.4.3 Hull Shader

First stage of Tessellation (Tessellation is for adding more details to an object). Tessellation is a process of dividing larger primitives in to small triangles depending on the level of detail (LOD).
A hull shader which is invoked once per patch transforms input control points that define a low-order surface into control points that make up a patch[2]. It also does some per patch calculations to provide data for the tessellation stage and the domain stage.

### 2.4.4 Tessellator

Tessellator is a fixed function stage initialized by hull shader to the pipeline. The major function of tessellator stage is to subdivide a domain into many smaller objects like triangle, points or lines. The tessellator also operates once per patch using tessellation factor i.e. it takes tessellation factor as an input. These tessellation factors were computed for each edge and the interior of each patch by hull shader.[2]

### 2.4.5 Domain Shader

The domain shader unit tracks the caching of shaded domain data. It also allocates and de-allocates memory resources for the domain data. The domain shader dispatches threads to shade domains that miss the domain cache. Lastly, Domain Shader also passes the vertex to Geometry Shader unit for further processing.

Properties of the domain shader include:[2]

- A domain shader is invoked once per output coordinate from the tessellator stage.

- A domain shader consumes output control points from the hull-shader stage.

- A domain shader outputs the position of a vertex.

### 2.4.6 Geometry Shader

The Geometry Shader (GS) fixed function stage corresponds to the DirectX10/11 Geometry Shader pipeline stage. It is an optionally enabled stage that can be used to convert each incoming object into a possible series of new primitive topologies. When enabled, the GS unit will extract independent objects from the incoming vertex stream. A GS thread will be spawned for each independent object within the topology. The thread can generate some number of new primitive topologies.The resulting vertex data is stored in GS-owned registry entries.

### 2.4.7 Resterizer

Rasterization is the process of finding all the pixels (picture elements) inside the triangle. Generally, we find all the pixels whose centers are inside the triangle and this is done by some specific rules called rasterization rule. The rasterization stage converts vector information composed of shapes or primitives into raster image which is composed of pixels for the purpose of displaying real-time 3D graphics on to the screen. During this stage each primitive is converted into pixels, while interpolating per-vertex values across each primitive.[2].In general,this unit is responsible for displaying 3-D shapes on a computer. I.e. a vector graphics format is converted to a raster image (pixels or dots) format.

### 2.4.8 Pixel Shader

A pixel shader is a functional unit which calculates effects on a per-pixel basis. So a pixel may be rendered, lit, shaded, and colored each frame depending on the resolution. So this unit generates more surface details allowing the user to see effects beyond the triangle level.

### 2.4.9 Pixel Output

This stage will display the output on the screen.

## 2.5 High volume manufacturing and test related Concepts

### 2.5.1 Introduction

In this chapter, we will discuss in detail the concept of High Volume Manufacturing (HVM). We will also discuss the test related concepts such as structural and functional testing, fault modeling, types of fault model, fault classification, fault coverage and the concept of fault simulation.

### 2.5.2 High Volume Manufacturing

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 in figure 2.6

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. Assembly:

   The packaging part of manufacturing is done in this phase.

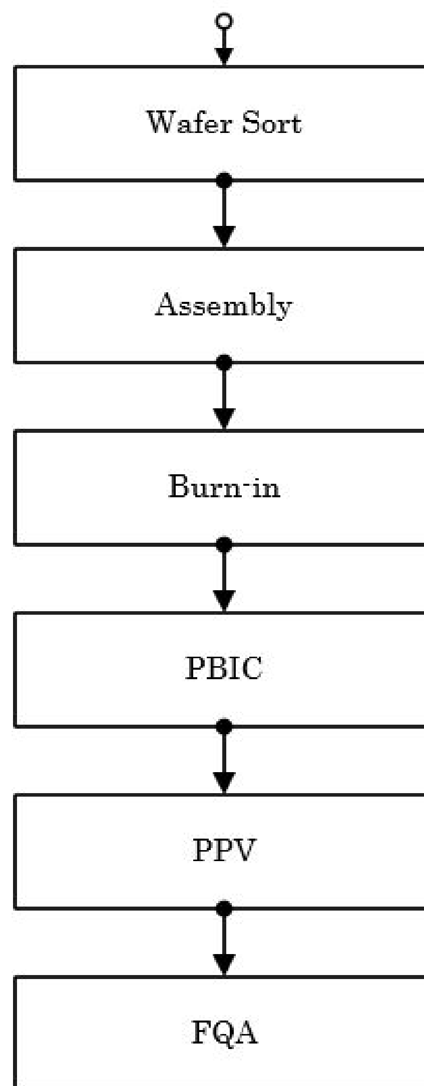



Figure 2.6: High Volume Manufacturing Flow[1]

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.

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.

   (1) Assembly related defects

   (2) Hot temperature sensitive defects

   (3) Fab process related defects

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) and also measure and validate the target product DPM against the corporate goal[1].

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:

(1) Units that have been damaged physically or electrically during the manufacturing flow.

(2) Units that were mis-binned due to a handler malfunction.

(3) Accidental mixing of failing units with passing units in production.

## 2.6 Structural vs. Functional Testing

Testing a VLSI system for faults can be divided into two categories. Functional and Structural. Functional testing is the testing done to verify the functional operation of the design. It checks that the design compiles to the functional specification of the chip, i.e. the implementation does what it is intendeddesigned to do. This testing is usually developed in the RTL design stage.
Structural Testing is the actual verification for proper construction of each element in the circuit. Fault Grading provides a tool for developing an effective structural testing test suite.
A complete functional test should test that all system functions are performed correctly under all possible input conditions. In theory, the test should examine the systems behavior in each sequence of combinations of input values. Consider a simple 32 bit adder, it has 65 inputs (two 32 bit inputs and one carry bit). To test this adder functionally, $2\hat{6}5$ tests are needed, which is a tedious task. So for a complex VLSI circuit, functional testing is much more complex. But if there is enough information on the structure of the system, a relatively small number of structural tests can be applied to detect a given set of faults. So structural testing saves time.

## 2.7 Fault Model

As the size of the VLSI circuits increased, it became difficult to test every potential failure in the circuit. In addition engineers faced problem of addressing actual physical defects when working with a logic model. The solution in the VLSI industry has been

to adopt a fault model which works in a similar manner to the logic model, i.e. the fault model maps physical defects to faults in a logic model[4]. The target of the fault model is to assist in developing tests that will detect actual defects when production tests are run and represent the highest percentage of all possible failures.

## 2.8 Circuit Defects and Fault Modeling

A VLSI chip may fail due to many reasons like design or fabrication flaws, environmental factors, or a combination of these. The resulting physical defects consist of break in lines ,shorts between lines at the dame interconnection level, shorts through the insulator separating different levels, shorts to substrate, point detects and imperfections such as scratches across the chip. Trying to test the actual physical defect is a very complex procedure. Studies have shown that an effective way to test circuit is to observe a circuits behavior by building a logic model and using the logic model to test the physical character of the circuit. These logic models used for testing physical characteristics are called fault models and are designed to test if the engineers tests will detect a defect that, if found in the actual chip, would cause the chip to be rejected.

## 2.9 Types of Models

The single stuck-at fault is the most common fault model presently used in the industry since it models the logical behavior of the most frequently occurring physical failures. The stuck-at model was developed for the designs represented in the logic gate level[4]. At gate level, it is assumed that, regardless of the actual technology in which the gates are implemented, most of the internal defects will be manifested as logic malfunctioning of the gates. Thus this model is assumed to have a high correlation between detection of logic faults on the logic gate pins, and detection of defects in the actual manufactured products.

There are other models such as connector-switch-attenuator (CSA) model, the stuck-open and stuck-short models, which were specifically developed for the transistor level, but these, are very complex and take more execution time.
The stuck-at model seems to be successful in covering a high percentage of defects, and hence it is widely used.

### 2.9.1 Stuck at model

The Stuck-at model assumes that, a fault can be modeled by one of the following two fault types,

1. Stuck-at 1 (S@1) - when a pin or signal is shorted to logic HIGH
2. Stuck-at 0 (S@0) - when a pin or signal is shorted to logic LOW

The stuck-at faults are applied to logic gates pins (if gates can be extracted from the circuit), and on transistor pins (if extraction of logic gates is not possible)

### 2.9.2 Types of S@faults

Following are some of the standard S@ faults

1. Input S@
2. Output S@
3. Node S@

The S@ models for input and node faults is simple. An input fault causes the input pin to stuck @ logic one or logic zero. It determines the value that goes into the gate, but has no effect on the signal previously driving the gate. A node fault affects all the inputs it drives, but the output it is driven by has no effect on it. In case of node driven by only one element, the output S@ is equivalent to the node S@. In case of

a multi driven node, the output fault affects only the node fanin connected to the element. Thus, contention can occur between the output of the faulty element and the other nodes fanin.
If the output fault is shorted to power or ground, other outputs driving the node does not have any effect. If the output fault is not shorted to power or ground, then the signal at the node can be a resultant value due to other drivers also. (Usually the former type of output fault is considered while modeling)

## 2.10 Fault Collapsing

Fault collapsing is the ordered process of using ordered relation among faults[4]. Two types of ordered relation among faults are

1. Equivalence relation:
   Two faults f and g are said to be equivalent, if any test sequence that detects test f will also detect fault g and vice versa. There are a variety of inserted faults that are equivalent. For eg: S@0 fault at NAND input is equivalent to S@1 fault at its output.

2. Dominance relation:
   Fault g is said to be dominated by a fault f, if every possible test sequence for fault g is also a test sequence for fault f. For eg: S@1 fault at NAND input is dominated by the S@0 fault on its output.

## 2.11 Fault Classification

Fault classification mainly given based on nature of the fault in any faulty microcircuits. And can be defined as below.

- Detected [DET]: A value opposite to the good machine value has been observed.
- Undetected [UND]: Good and faulty machines have the srame 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.

## 2.12 Fault Coverage

Fault Coverage is a quantitative measure of tests grading. It is defines as the ratio of number of faults detected by a test to the total number of detectable faults. It is always expressed as percentage and can be explained using the equation.

$$FaultCoverage(\%) = \frac{Number\ of\ faults\ detected\ by\ test(s)}{Total\ Number\ of\ detectable\ faults} * 100 \tag{2.1}$$

Where Total Number of detectable faults = Total faults - Un-testable faults.

## 2.13 Concept of Fault Simulation

Fault simulation is known as the mechanism of evaluating and grading test completeness by modeling faults and measuring the effect on circuit behavior[3]. During fault simulation, faults are injected into a defect free version of the circuit, and checks whether the test vectors can detect any 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 considered to be detected by the test. This means that, if a test detects a fault, then the same test, when run as a production test, should find a faulty chip and reject it, i.e. fault simulation allows emulation of a production test. So it can be considered as the mechanism of evaluating and grading test completeness by modeling faults and measuring the effect on circuit behavior.
Following block diagram explains fault simulation process

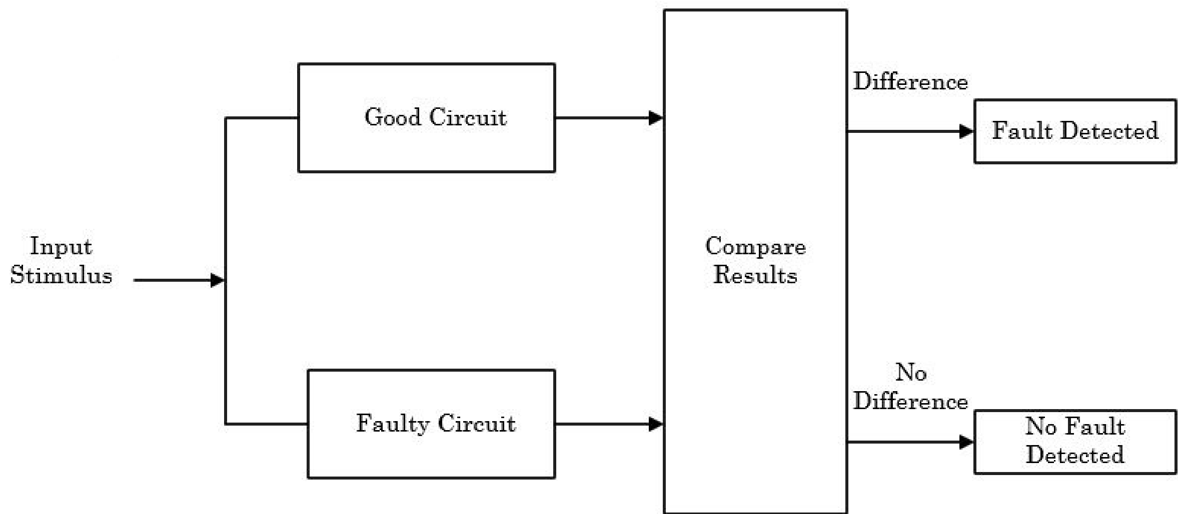



Figure 2.7: Fault Simulation Process

Fault simulation helps in developing high quality manufacturing tests and thus reducing the number of defective parts shipped to the customer. Also by analyzing the results of fault simulation test engineer can eliminate redundant or ineffective tests to achieve a better set of tests.
Fault simulation is the testing step of the Fault Grading process.

## 2.14 Fault Simulation Algorithm

To evaluate the faults there are various algorithms are elaborated as below.

### 2.14.1 Serial Fault Simulation Algorithm

Serial fault simulation is most 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[5].

Figure 2.8 represents the following algorithm which is used in this project.

1. Simulate fault-free circuit and save responses.

2. Repeat the following steps for each fault in the fault list:

   (1) Modify netlist by injecting one fault

   (2) Simulate modified netlist, vector by vector, comparing responses with the saved responses.

   (3) If response differs, report fault detection and suspend fault simulation of remaining vectors.

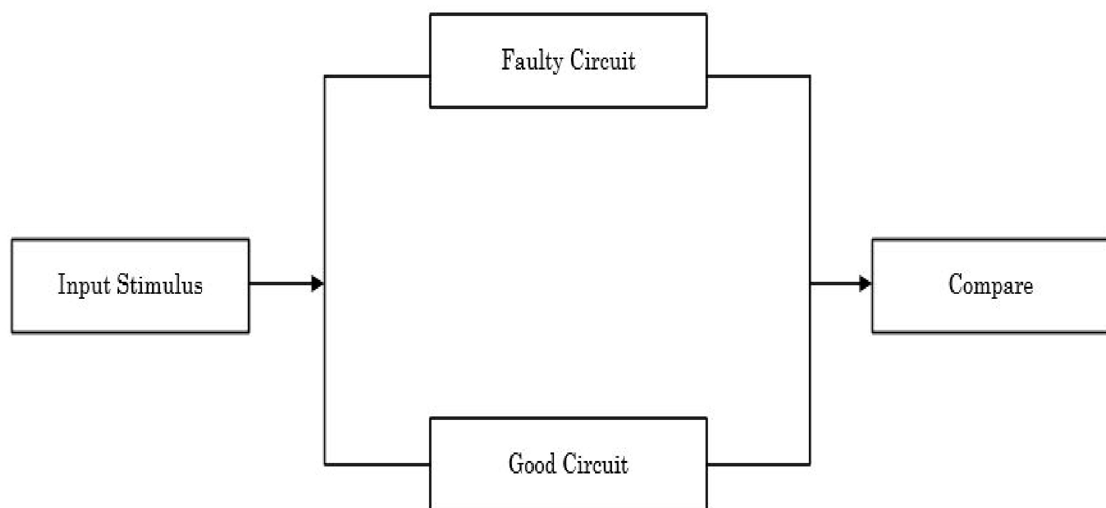



Figure 2.8: Serial Fault Simulation Algorithm

Advantage:

1. Easy to implement, needs only a true-value simulator, less memory.

2. Most faults, including analog faults, can be simulated.

Disadvantage:

1. Much repeated computation

2. Most inefficient

### 2.14.2 Parallel Fault Simulation Algorithm

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.
This algorithm is shown in figure 2.9

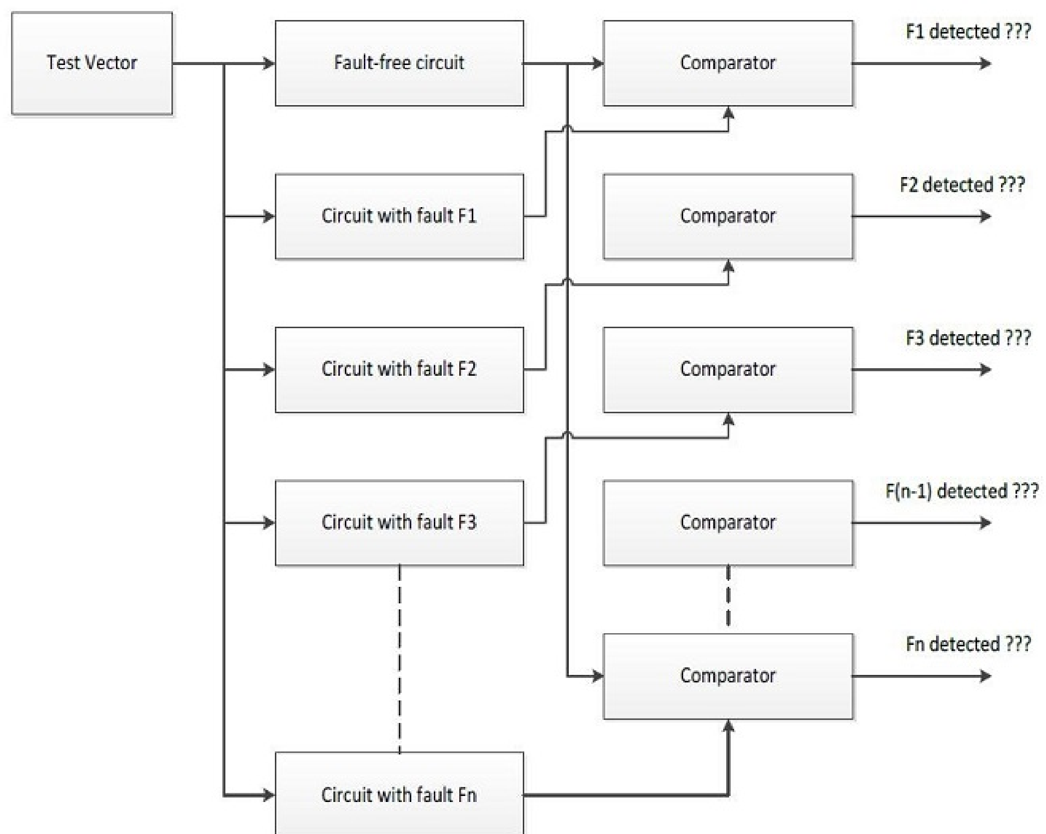



Figure 2.9: Parallel Fault Simulation Algorithm

This algorithm is more efficient then serial fault simulation algorithm.but the disadvantage is, it is difficult to implement.

### 2.14.3 Concurrent Fault Simulation Algorithm

Figurre 2.10 represents the Concurrent fault simulation algorithm. It is the most widely used fault-simulation algorithm and takes advantage of the fact that a fault does not affect the whole circuit[5]. Thus, 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 same copy of portion of that circuit. So, 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 allows 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. So, 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 appropriately.

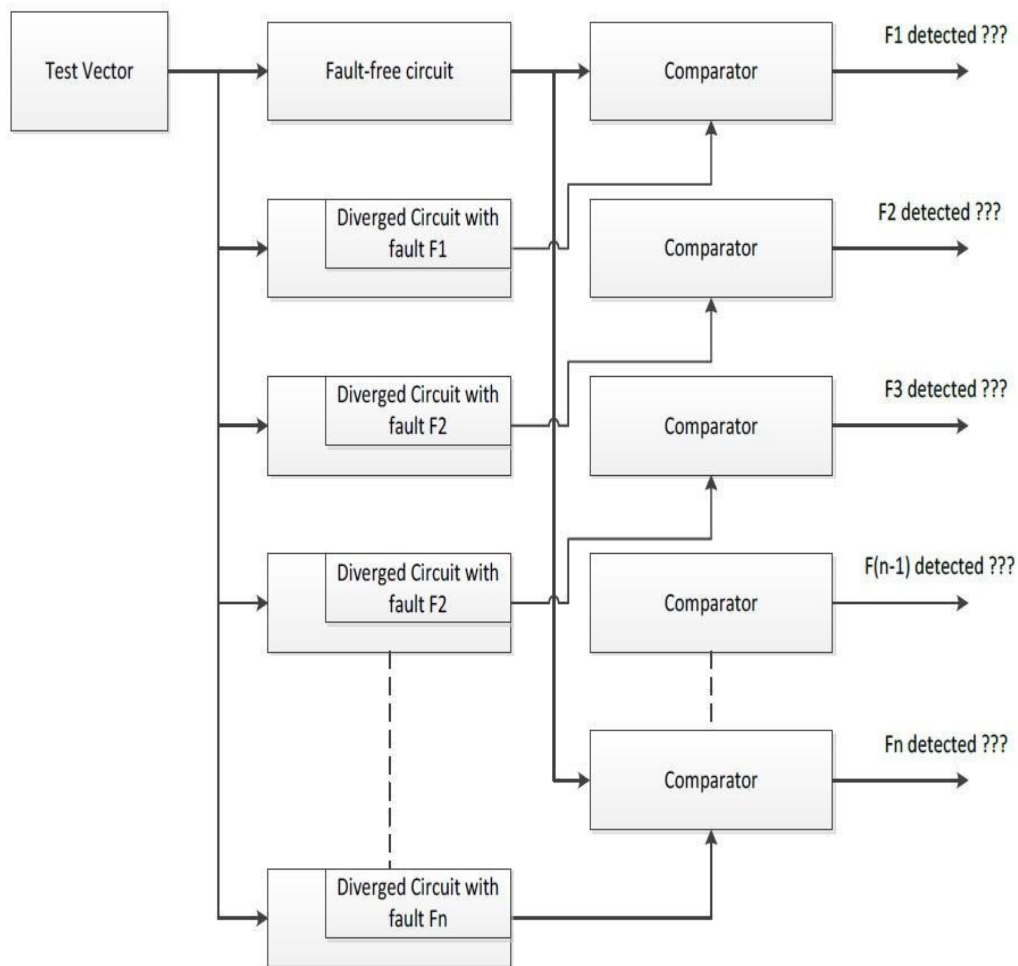



Figure 2.10: Concurrent Fault Simulation Algorithm

This algorithm is faster then any othe algorithm. So, it gives faster results as it covers only a part of circuit.But on the cost of memory consuption.

## 2.15 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.

## 2.16 Fault Detection

The tests written for detecting the faults are simulated by the fault simulation tool on the fault model to ensure all faults are detected. There are nodes in RTL design which allows the tool to inject the test vector and observe the propagated faults, known as control point and observation point.The fault simulation tool will compare the outputs of the good model and 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).Given below, an example of how a fault is detected.

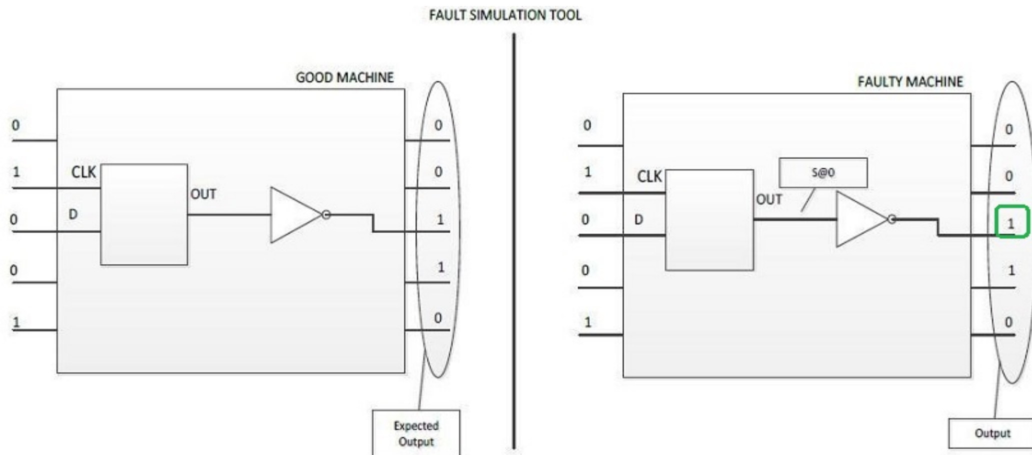



Figure 2.11: Example showing how a test vector cannot detect a fault

Figure 2.11 shows the outputs of the good and faulty machines on the test vector 01001. With this input the faulty machine as well as the good machine gives same output. Hence the fault simulation tool is not able to detect the stuck at fault present at the input of the inverter using the test vector 01001.

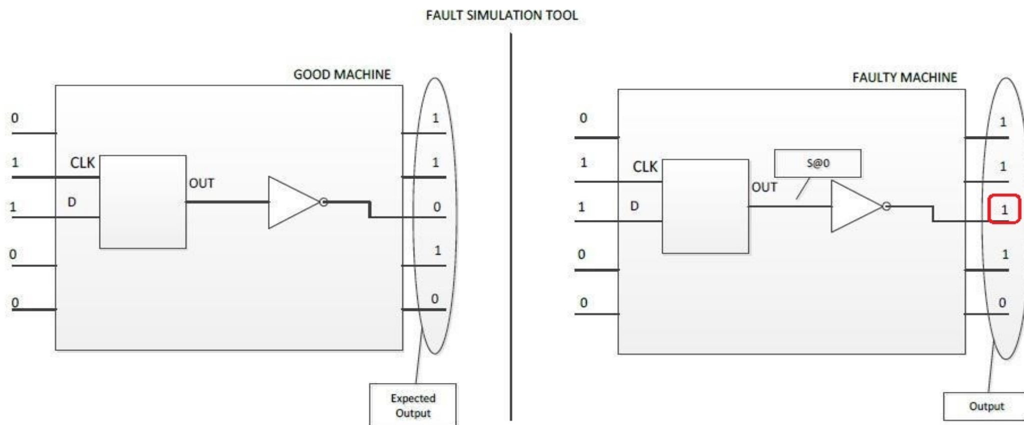



Figure 2.12: Example showing how a test vector detects a fault

Figure 2.12 shows the outputs of the good and faulty machines on the test vector 01100. With this input the good machines output is 11010 and the faulty machines

output is 11110. So by comparing, the fault simulation tool will detect the stuck at fault present at the input of the inverter. So a test which injects the test vector 01100 will be able to detect a S@0 fault at the input of the inverter, and hence the test is considered as an effective test.

## 2.17 Summary

In this chapter we saw basics of graphics and 3D graphics in brief. To validate any digital circuit, we have to learn about types of faults as well as the evaluation process for make circuit fault free. The major step for fault evaluation is Fault simulation. Fault simulation is the mechanism used for evaluating and grading the effectiveness of a test. The result of fault simulation gives an idea on how well the test vectors detect certain manufacturing defects. Serial fault simulation algorithm is used in the project.

# Chapter 3

# Fault Grading

## 3.1 Introduction

In this chapter, we will discuss in detail the Fault Grading methodology. We will also discuss the fault grading methodology followed to develop test content for Inte's advanced Graphics processor.

## 3.2 Fault Grading Methodology

Fault Grading (FG) is the entire process of deriving a series of tests, and evaluating/grading their effectiveness in detecting possible manufacturing defects[6]. FG has two purposes, one to serve as a process for test development, and the other to measure and improve the quality of production tests. Fault grading is implemented by performing fault simulation, the mechanism for evaluating and grading test completeness. The detailed FG flow is shown in the figure 3.1.
Sampler and execution units are the units which are fault graded to meet the DPM requirements. The steps that are followed is shown in the block diagram.

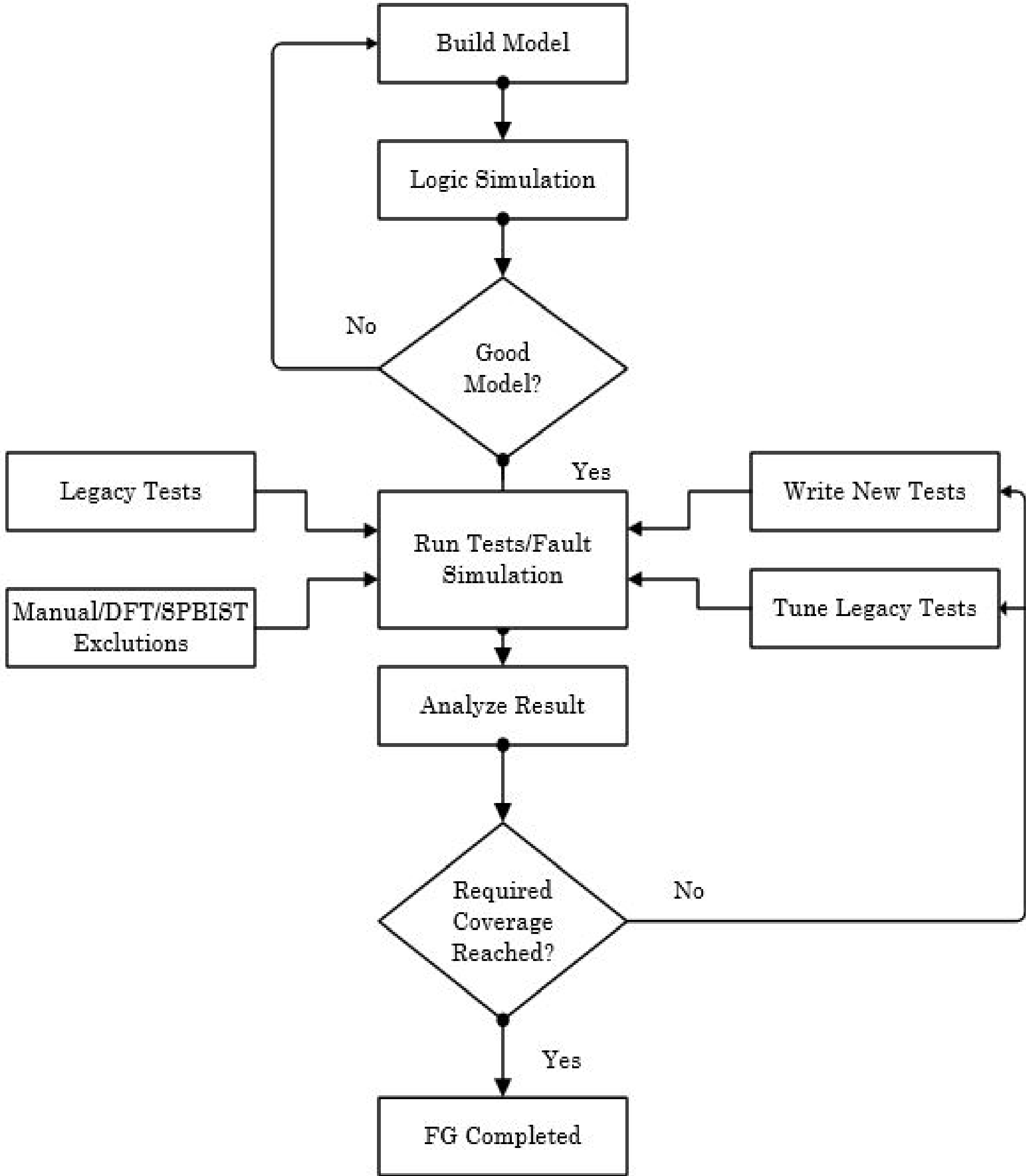



Figure 3.1: Flow chart of fault grading[7]

### 3.2.1 Build Fault Model

The first step is to build the fault model for the RTL Design. This model will be covering sampler unit of the GPU

### 3.2.2 Running all Legacy tests on this model

On this model all the legacy tests are run and the status of the faults detected is changed to DET (Detected).Legacy tests are tests which are inherited from the previous generation of the processors. There might be compatibility issues with the legacy tests.These issues need to be resolved and maximum number of legacy tests should be enabled. Hence debugging of failing tests becomes very important.

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.

The simulator calculates the fault coverage obtained by the legacy tests using the formula mentioned in equation (2.1).

### 3.2.3 Find good tests from legacy tests

Find good tests from legacy tests and modify them so that more faults can be detected. For example in the case of decoder and encoder unit, unit wise analysis was done to check which units lack coverage. Then the good tests pertaining to that logic were modified to target the unit. This helped in increasing the number of faults detected.

### 3.2.4 Implement Exclusions

According to the guidelines for effective fault grading , the un-testable faults, DFT related logics and BIST logics are excluded. This is called the method of exclusion. If the required fault coverage is not yet met,then write new tests or tune Legacy tests.

### 3.2.5 Write new tests

The areas in the design having less coverage are found out and the tests are written specifically targeting these areas. Less coverage areas can be obtained by analysis the fault coverage results obtained from the above steps.

## 3.3 Randomized test generation

The sampler and execution unit functionality involves extensive mathematical computation. Writing test cases to cover all the scenarios becomes exceedingly difficult and a tedious process. Covering all the corner cases using limited set of test cases becomes very difficult. Hence there is a need to randomize the test cases in order to hit the corner cases.
In this process, the usage of the randomized test generation tool was studied. The good coverage yielding tests were selected and randomized using the randomization tool. These randomized tests were run on the sampler pipeline.
This process helped to improve the coverage significantly in various units across the encoder and decoder pipeline.

## 3.4 Test Execution Flow

The below figure 3.2 shows the basic flow of test execution applied for sampler unit. The first step is obviously the requirement of good test.There are basically two types of tests has been used in intel's environment.

- DYN Tests: These are strictly typed and only directed tests.we use them as legacy tests.
- GRITS(Graphics Random Instruction Testing) tests: These tests are ruby based with GRITs API.These tests interoduces randomness and allows us better flexibiity to modify them

The test flow execuation of the GPU is shown as below.

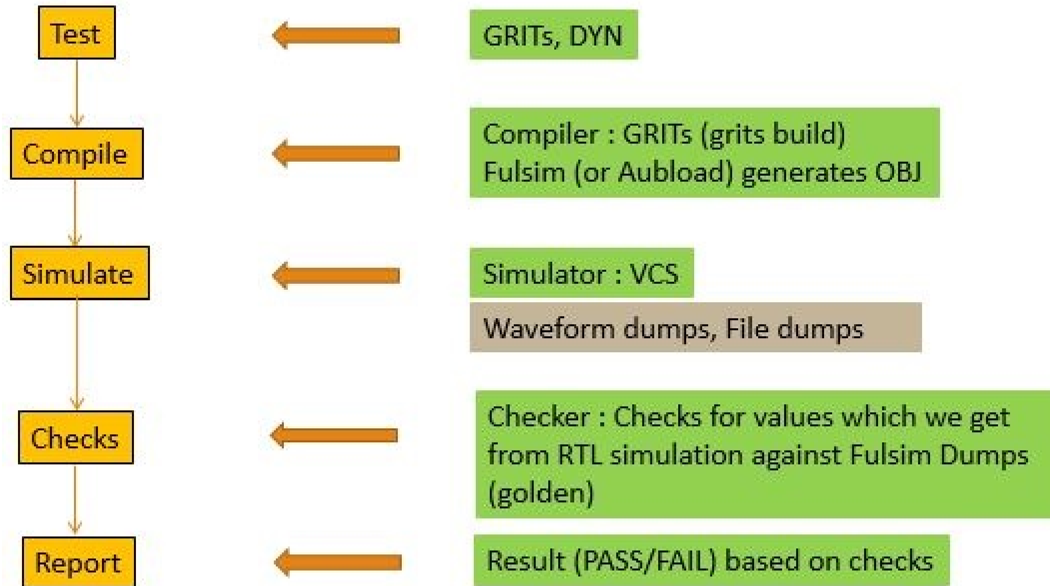



Figure 3.2: Flow of test execution

In the process of execution after test has been generated , it goes to compilation phase.In this stage compiler will checks the test format,include files and generates an object file into dump area. So, now after compiling test, it will now simulate in VCS simulator. During the simulation it will dumps various files like runsim.log.gz,vsim.log.gz and various wavwforms as well as .rpt and .fsdb files into dump area.Now moving further , checkers checks and compares to dumps, one we get from RTL simulation and fulsim dumps which already available in golden area. At last, the final result based on checks will be stored in a log file named 'runsim.log.gz' insode the regress path.

## 3.5 Test Debug

Debugging the test failures is one of the critical phases in the fault grading flow. When the test fails, we need to root because the failure by checking the error files.

### 3.5.1 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.

### 3.5.2 Run limit reached

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.

### 3.5.3 L3-Loader Fit-ability Issues

This issue arises when any big image or data processed inside the l3 memory of GPU and memory can not process such a big data. So, in this case we try to keep image small or reduce data to be processed and try running them again. We also use some compression options available in run commamnd to compress the big image with predefined ratio. This issue is most occuring among all other issues for sampler unit.

## 3.6 Node Observability Architecture(NOA)

The tests written for detecting the faults are simulated by the fault simulation tool on the fault model to find out the number of faults are to be detected. NOA is a

DFX concept intorduced to help the design engineers to track the internal signals of the design. So, this methodology have made simple allowing the RTL designers suggested critical signals status captchered to the registers at a certain time. We use this NOA concept to cover some of the RTL area where the NOA capturing signals come in to picture. We use a method where high toggling signals from a test will be programmed to the NOA registers. So,this way the NOA path where the coverage was lagging will be covered.

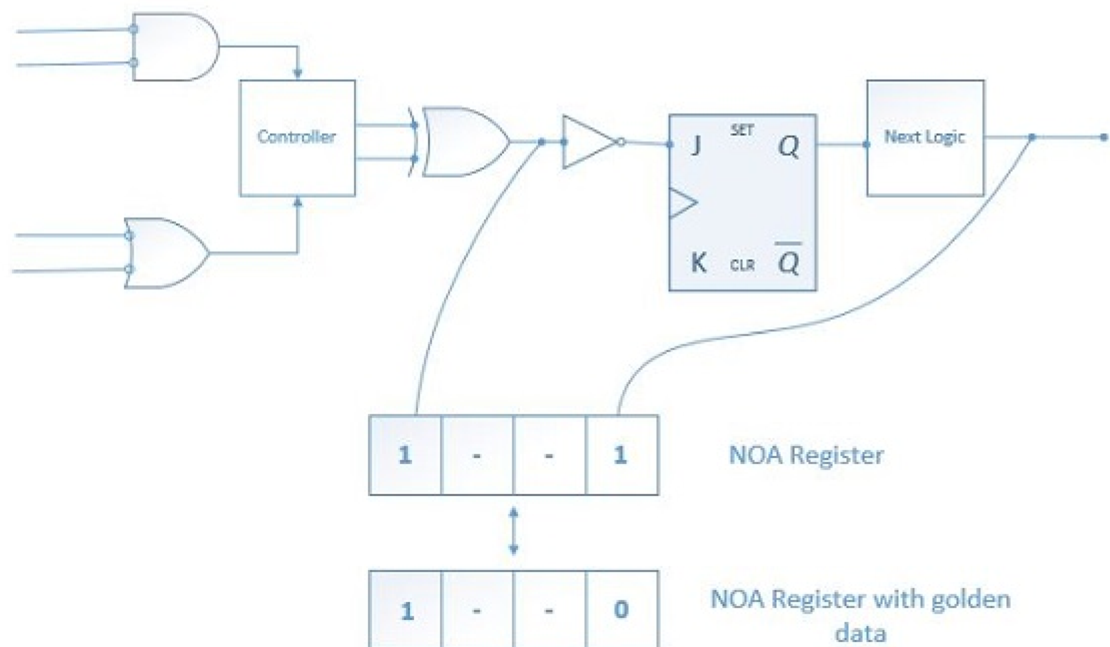



Figure 3.3: Working principle of NOA

The working of this method can be understood by the following example. So, let us consider one small functional unit inside the whole circuit as shown in Figure 3.3. According to this diagram there are two different nodes where we are targating the faults to be detected. So, by enabling NOA register, we are able to get output of each of the targeted nodes to a register called as NOA register. We have a golden database for referance to compare each of the outpus. Now after getting output from those

nodes into NOA register, it will be compared bit by bit with golden NOA register and any mismatch will be counted as fault detected.
So, like wise by using NOA we can hit more number of detects in uncovered areas of the circuit.

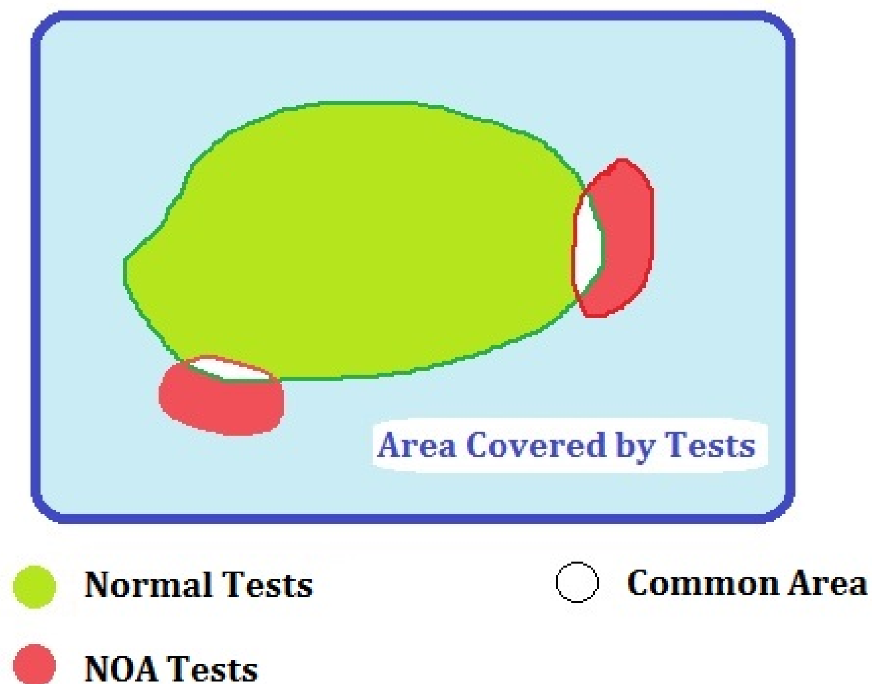



Figure 3.4: Coverad area with NOA Tests

The NOA methodology has various benifits which can be breifly classified as in Figure 3.4. Where the green portion is the area hit by the normal tests without including NOA ports and the red portion is the area hit by including NOA ports. There is possibility for NOA tests that they can hit already detected areas as well as uncovered areas in the circuit. So, repeated detets can be ignored and new detets can be considered for calculation.

## 3.7 Summary

In this chapter the Fault grading methodology was disccused for the Intels Graphics processor.Also, there are various issues occuring while validating a apecific unit of GPU has mentioned. In adddition with NOA efforts we are able to get close number to the required target. In this project we wish to achieve the overall fault coverage of around 81% of sampler unit of GPU.

# Chapter 4

# Results and Analysis

In this chapter, we will see the tools that are used to track the fault grading activity on sampler pipeline tests.we will also discuss about run command by which the results has been obtained.

### 4.0.1 Regression Results

Figure 4.1 shows the block diagram of the validation flow. The test vectors are generated and injected into the RTL and the golden reference model.

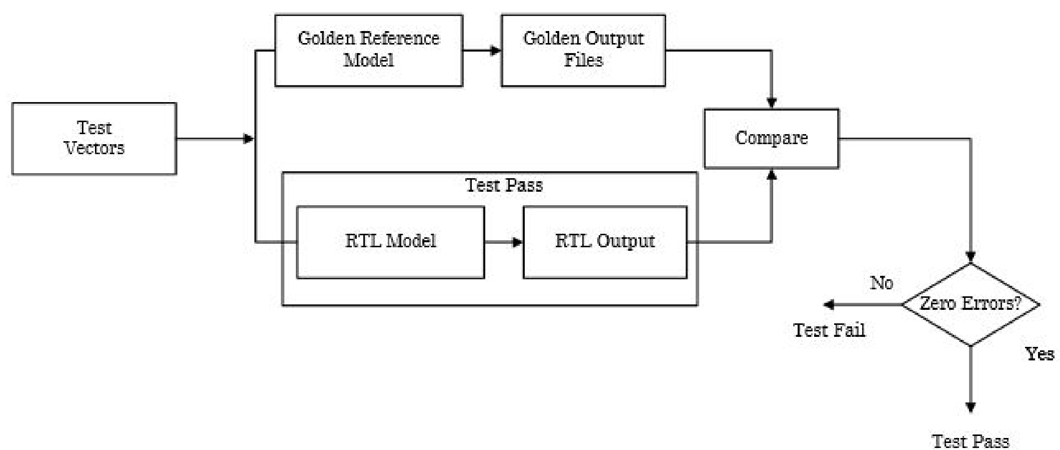



Figure 4.1: Validation Flow

The outputs of both the models are compared and the error is reported if there is a mismatch.

### 4.0.2 Run Command

Run Command is the command used to run the test. Tests can be run in 2 ways, depending on the number of tests that need to be run. If a set of tests are needed to be run then we launch the run in netbatch mode, whereas if it is a single test we run it on a local machine in the normal mode.

1. Netbatch mode Netbatch mode is used to run a group of tests defined in test list. The tests submitted to netbatch are sent to a particular wait queue on remote machine. Their run progresses as they reach the top of the wait queue and depending on the availability of free machines in the pool.

   The command used to run tests in netbatch: " runreg -l testlist (command)".

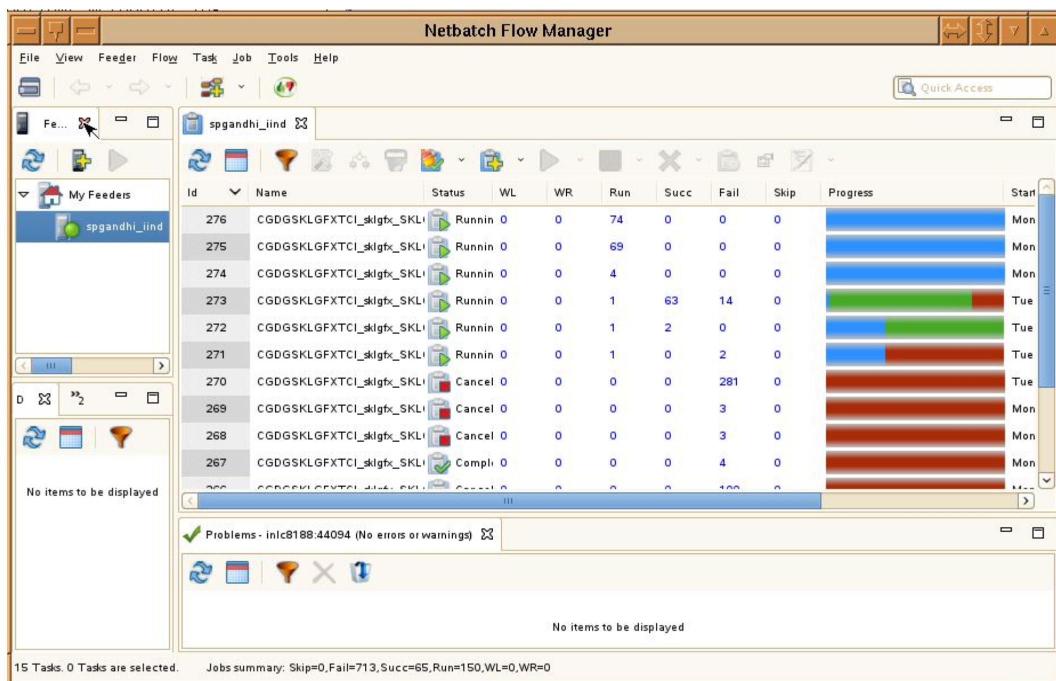



Figure 4.2: Netbatch flow manager GUI

The netbatch flow manager tool is used to monitor the status of regressions. When we invoke the netbatch manager GUI shown in Figure 4.2 pops up. It shows the current status of running test list.It also gives information about the failling and passing no. of tests from the test list.

2. Local interactive mode Local interactive mode is used for running a single test interactively without having to wait through the netbatch queue.In this run we can directly hit command on terminal and the processos comes on the same terminal on the same machine. The command used to run a test in local mode: "runsim -t testlist (command)".

The netbatch manager tool is used to monitor the status of regressions. When we invoke the netbatch manager GUI shown in Figure pops up. It shows the pass rate and error types.

### 4.0.3 Fault Coverage Result of Sampler unit

Figure 4.3 shows improvement acheived for the fault coverage of the sampler unit after each of the steps followed in the fault grading methodology. At the start time of the validation process there are all faults Undetected and hence it sarts from 0% fault coverage of sampler unit.After running some basic tests we got some DETs in samplar so it got improved. So, to improve further or find more DETs in particular unit, we have to simulate more no. of tests on it. By running legacy tests and some unit level tests, the overall fault coverage has improved. We also randomize the high DETs giving tests to hit more no. of gates. In addition by NOA programming we enabled NOA registers and with the help of this effort we reached close to required coverage target. So,the overall progress with the help of all possible efforts is mentioned in above graph.
Where on x-axis,

A: At the start

B: Basic Tests

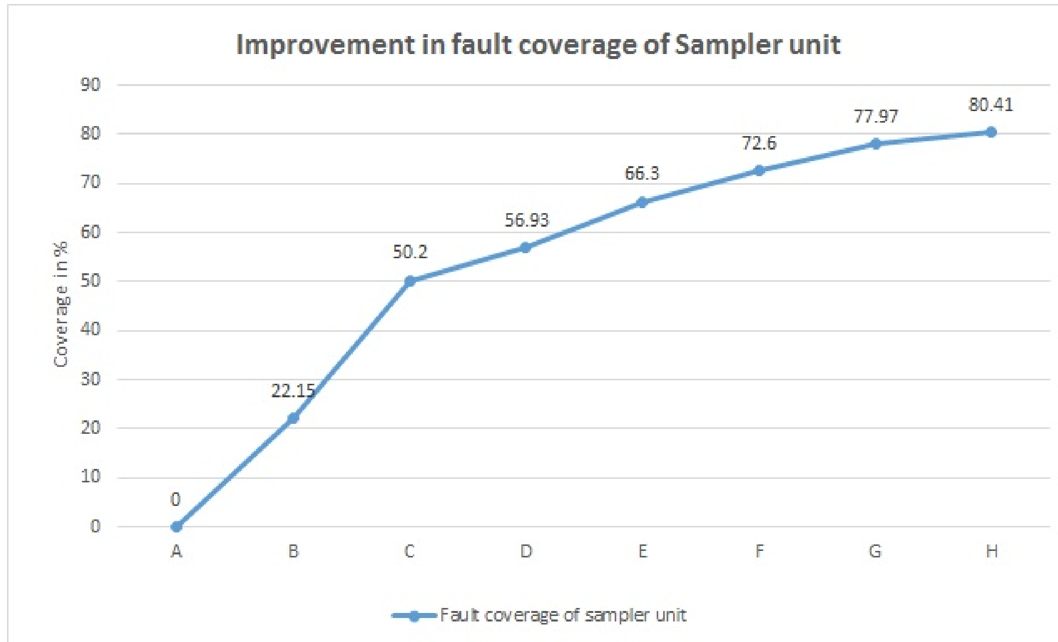



Figure 4.3: Graph showing Fault Coverage progress.

C: Legacy Tests

D: Modified Tests

E: Unit level tests

F: Excluding DFT logic

G: Randomized Tests

H: NOA enabled tests

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

## 4.1 Summary

This chapter explains the results acheived during the project work.There is a tool to extract the informaton about the test status in netbatch. By using a methodology on fault grade we acheived 80.41% as current fault coverage.

# Chapter 5

# Conclusion and Future Scope

## 5.1 Conclusion

This project work came up with a methodology to develop test content to meet the desired fault coverage target. It already been explained in detail the various steps in the methodology and how they can be implemented to achieve the desired fault coverage. Through out this project work by the help of the fault grading we found out the areas in the design which are lagging in coverage and to escalate those we tried to 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. These tests are supposed to give the same fault coverage, obtained in the FG process using fault model, when simulated on actual silicon. This will help in making sure a low DPM of shipped products. The methodology can be used in future products for the efficient test development activity.

## 5.2 Future Scope

During the project the analysis was done regarding the inability to detect the faults in some logics. This revealed the lack of observation points in some logics. Thus the future scope of the project would be to add more observation points in the design so that it can detect all the faults and can improve coverage beyond the target.

# References

[1] Intel Documents

[2] Dennis Crain, Dale Rogerson. "*Graphics pipeline*". Internet: http://msdn.microsoft.com/enus/library/windows/desktop/ff476882%28v=vs.85%29.aspx, Jun.08,2014 [Aug.23, 2014].

[3] Yu Zhang and Vishwani D. Agrawal,"*An Algorithm for Diagnostic Fault Simulation*",IEEE Conference Publications 1993,pp-144-147.

[4] Jacob A. Abraham and W. Kent Fuchs, "*Fault and Error Models for VLSI*", Proceedings of IEEE, Vol 74, No 5, May 1986

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

[6] Weiwei Mao and Ravi K. Gulati ,"*Improving Gate Level Fault Coverage by RTL Fault Grading*" , International Test Conference, 1996, Paper 6.3, pp 150-159

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

# Appendix A

# 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. Primary input: The input to the model is called as the primary inputs. Primary inputs to the logic model are the only controllable inputs.
8. Primary Output: The output from the model is called as the primary output. Primary outputs are the only directly observable outputs from the logic model.
9. Good Machine: Good machine is a defect free copy of the circuit used as the basis for comparison during fault simulation
10. Faulty Machine: Faulty Machine represents the behavior of the circuit on which a fault has been inserted
11. Test Grading: Test grading involves the use of fault grading to test the effectiveness of a test in detecting the defects.
12. Pre Silicon validation: Validation done on RTL model to detect design faults.
13. Post Silicon Validation: Validation done on actual silicon for faults.