DESIGN FOR TESTABILITY – PATTERN GENERATION AND SIMULATION OF COMPLEX CIRCUITS

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

MASTER OF TECHNOLOGY

in VLSI Design

Submitted By

AMANDEEP KAUR

Roll No. 601762002

Under Supervision of

Dr. Sumit Vyas

Designation- Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING

DEPARTMENT THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY

(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

JULY, 2019

DECLARATION

I, **Amandeep Kaur** hereby declare that the work presented in this project " **Design for Testbility – Pattern Generation and simulation of Complex Circuits**" in completely fulfillment of the requirement for the award of degree of **Master of Technology (VLSI Design)** submitted at **Electronics and Communication department**, Thapar Institute of Engineering & Technology (Deemed to be University), Patiala is an authentic record of work carried out under supervision of **Dr. Sumit Vyas (Assistant Professor, ECED, TIET)** from June 2018 to May 2019.The matter presented in this has not been submitted either in part or full to any other university or institute for the award of any other degree.

Signature of Student

Dated 15/7/19

Dr. Sumit Vyas
Assistant Professor
ECED, TIET Patiala

CERTIFICATE

This is to certify that **Amandeep kaur (601762002)**, a student of M.E. (VLSI Design), **Thapar Institute of Engineering and Technology, Patiala**, has successfully completed one year internship program (4th June,2018 – 30th May,2019) at **STMicroelectronics, Greater Noida**. Her title of dissertation is **"Design for testability pattern generation and simulation for complex circuits"**.

I wish her success in life.

Date: 29|05|2019

Manish

Manish Sharma

Senior Staff Engineer

STMicroelectronics, India

Greater Noida

ACKNOWLEDGEMENT

The internship opportunity I had with STMicroelectronics Pvt. Ltd. was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period. I express my deepest gratitude and special thanks to Mr. SABYASACHI DAS (Group Manager) who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep me on the correct path and allowing me to carry out my project. I express my deepest thanks to Mr. MANISH SHARMA and Mr. ARJUN SINGH for mentoring me throughout the internship. And I am also very thankful to Dr. SUMIT VYAS from Thapar institute of engineering and technology, Patiala who have guided me throughout and also helped me in learning the VLSI concepts in a better way, also I would like to say thanks to Dr. ALPANA AGARWAL (HOD) who have always stood by me and helped me to bring out the best in me. I am also very grateful to Mr. DEEPEN S.TALATI for helping me in learning various basics of DFT and scripting. I am thankful to all the members of DFT Team at STMicroelectronics Pvt. Ltd for their efforts in making me achieve my goal to be a better design engineer.

ABSTRACT

Even though a circuit is designed error-free, manufactured circuits may not function correctly. Since the manufacturing process is not perfect, some defects such as short-circuits, open-circuits, open interconnections, pin shorts, etc., may be introduced. Points out that the cost of detecting a faulty component increases ten times at each step between prepackage component test and system warranty repair. It is important to identify a faulty component as early in the manufacturing process as possible. Therefore, testing has become a very important aspect of any VLSI manufacturing system. Design for testability presents effective and timely testing of VLSI circuits. BIST architecture is used to test the circuits effectively compared to scan based testing. In built-in self test (BIST), on-chip circuitry is added to generate test vectors or analyze output responses or both.

TABLE OF CONTENTS

ABSTRACT iv

LIST OF FIGURES ix

CHAPTER -1 1

INTRODUCTION 1

1.1 OVERVIEW 1

1.2. FAULT CLASSES 2

i) Untestable Fault (UT): 2

ii) Testable Fault (TU): 2

iii) ATPG Untestable (AU) 3

iv) Undetected (UD) 3

1.3. TERMINOLOGIES AND PROCEDURE 3

(a) SCAN INSERTION 3

i) Scan Cell 3

ii) Scan flip flop 4

iii) Scan Chains 5

iv) Scan Clocks 7

(b) AUTOMATIC TEST PATTERN GENERATION (ATPG): 8

FAULT REPORTING 8

i) Test Coverage *8*

ii) Fault Coverage: *9*

ATPG Effectiveness: 9

Fault Collapsing 9

1.4. METHODS OF PATTERN GENERATION IN ATPG 10

i) Deterministic Pattern Generator: 10

ii) Random Pattern Generator: 10

iii) External Test Pattern Generation: 10

1.5. TEST TYPES 11

i) Functional Tests: 11

ii) IDDQ Tests: 11

1.5.1 IDDQ testing classified according to the CMOS circuits circuitry: 12

iii) AT-Speed Tests: 12

1.6. ATPG APPROACH 12

ii) Fault propagation: 13

iii) Justification: 13

(c) SIMULATIONS 13

i) COMPILATION: 13

ii) ELABORATION: 13

iii) SIMULATION: 14

SIMULATION BASED ON THE TIMING CONSTRAINTS: 14

i) No Time 14

ii) Best Case 14

iii) Worst Case 14

CHAPTER -2 16

LITERATURE SURVEY 16

CHAPTER -3 20

PROBLEM STATEMENT 20

CHAPTER – 4 24

WORK DONE AND SIMULATIONS 24

i) SCAN INSERTION 24

ii) AUTOMATIC TEST PATTERN GENERATION 29

iii) SIMULATIONS 39

CHAPTER -5 45

ISSUES OBSERVED AND RESULTS 45

CHAPTER – 6 47

CONCLUSION AND FUTURE SCOPE 47

CHAPTER -7 48

REFERENCES 48

LIST OF FIGURES

Figure 1: BASIC SCAN CELL 4

Figure 2: SCAN FLIP FLOP 5

Figure 3 SCAIN CHAIN 6

Figure 4 SCAN GROUP 7

Figure 5 FLIP FLOP SHOWN WITH A CLOCK........ 8

Figure 6 CMOS INVERTER WITH Vdd =0 AND NON ZERO CURRENT........ 11

Figure 7ASIC DESIGN FLOW........ 23

Figure 8 LOG FILE GENERATED AFTER SCAN INSERTION PROCESS 28

Figure 9 LOG FILE GENERATED AFTER SCAN INSERTION PROCESS 29

Figure 10 SHOWS HOW THE DATA IS TRANSFERRED IN PROCEDURE SHIFT........ 33

Figure 11 PROCEDURE SHIFT PROCESS........ 34

Figure 12 DATA SHIFTING IN LOADING AND UNLOADING PROCEDURE........ 35

Figure 13LOG FILE AFTER ATPG [1] 38

Figure 14 LOG FILE AFTER ATPG PROCESS [2] 38

Figure 15 LOG FILE AFTER ATPG PROCESS [3] 39

Figure 16 LOG FILE GENERATED AFTER COMPILATION 41

Figure 17 ELABORATION COMPLETED WITH ZERO ERRORS 42

Figure 18 SIMULATION COMPLETED WITH ZERO ERRORS 43

Figure 19 THE WAVEFORM SHOWING THE SIMULATED PATTERNS 44

CHAPTER -1

INTRODUCTION

1.1 OVERVIEW

The embedded system is an information processing system that consists of hardware and software components. The embedded system is now a day's present in many applications including Military applications, Automotive, Telecommunications and many more. We live in an era where the technology is increasing rapidly. We have risen from 1 to 100 transistors per chip to millions of transistors in a single chip. The growth is going on at a steady rate in different fields. With the growth in technology, the chip sizes are decreasing and complexity of the design is increasing. As the complexity increases, the chances of error are also increased.

The increase in design complexity has also made it difficult to access and diagnose the internal nodes, which may possibly have fault. So, it is important to have more efficient systems and designs which can help us check the errors in design. One way is to make a testable system design for a complex hardware/software system that function reliably throughout their operational lifetime. Now, testability can be defined w.r.t a fault. Now, a fault is considered testable when there exists a procedure to expose the fault, and the procedure then is used to test it using the current technologies. To test a circuit, first condition is that the fault must be approachable so that it can be diagnosed easily. So, we need technologies or methods which could increase the controllability and observability of a circuit.

In Design for testability (DFT), we come across such methods or procedures which increase the controllability and observability of a design. By **controllability**, we mean that the difficulty in setting the internal nodes output equal to 1 or 0 by setting the Primary input values. By **observability**, we mean that the difficulty of propagating the primary input values to output. The node in DFT is said to be testable if it has high controllability and observability.

Moore's Law: Moore's law is the observation that the number of transistors in a dense IC doubles about every year. It is an observation or historical trend and not a physical or natural law.

1.2. FAULT CLASSES

These are divided based on how the faults were detected or how they couldn't be detected.

i) Untestable Fault (UT):

For this no pattern could be generated to detect them. They do not cause functional failures in the device and are not included while calculating the test coverage. They are excluded by the tool in the beginning during ATPG and are given the appropriate categories as defined below.

a) Unused(UU) –These include all the faults which are unconnected to any circuit observation point.

b) Tied (TI) – In this the fault site is either tied to a value or tied to the identical fault site.

c) Blocked (BL) –It includes all the faults by which tied blocks all the paths to an observable point.

d) Redundant (RE) –These are those faults which are considered undetectable by the test generator. The test pattern generator performs the analysis for the faults which are undetectable under any condition which are referred as redundant faults.

ii) Testable Fault (TU):

These include all faults which cannot be said untestable.

a) Detected (DT) – These include all the faults which can be detected in the ATPG Process.

b) Posdet (PD) – These include all the faults which are said possible detected and not hard-detected. The possible detected faults will result into 0 or 1 after simulation.

iii) ATPG Untestable (AU)

These include all the faults for which test generator is not able to find or create a test pattern. These faults are due to some constraints or limitations on ATPG Tool. They can be shifted to possible detected faults if you remove the constraints.

iv) Undetected (UD)

This type of faults includes all those which are not under ATPG untestable or testable.

a) Uncontrolled (UC) –These have never achieved the value required at the fault required for fault detection. They are uncontrollable.

b) Unobserved (UO) – the faults which cannot be propagated and are unobservable.

1.3. TERMINOLOGIES AND PROCEDURE

We have number of faults in the design; to test the circuit for faults we need some steps as described below.

(a) SCAN INSERTION

There are different terminologies used under scan insertion –

i) Scan Cell

A scan cell is the fundamental unit in scan chain. It can be thought of a box which has an input line, output line and a circuitry inside it which will give the procedure for input data to travel towards output data. Each scan cell is associated with a memory element. With the scan-in input we give the test pattern to the design and then this test pattern is obtained at the output of scan cell i.e. at scan out.

Figure 1: BASIC SCAN CELL

ii) Scan flip flop

It is the basic D flip flop with the addition of some more pins which make it a scan flip flop. It has an input pin ‘D’, another input pin ‘scan input’ , and a select line ‘scan enable’ with an output.

When the Scan enable = 0, then normal functioning of the flop is there i.e. whatever input is given at D goes to the output at the clock pulse.

When Scan enable = 1, then test mode is ON i.e. scan input is enabled and the test data fed through the scan input pin goes to the output at the clock pulse.

Scan flip flops are used in the scan chains and when the scan enable goes to 1 the test mode is activated, which means now we will shift in the patterns through scan input and then capture the required data.

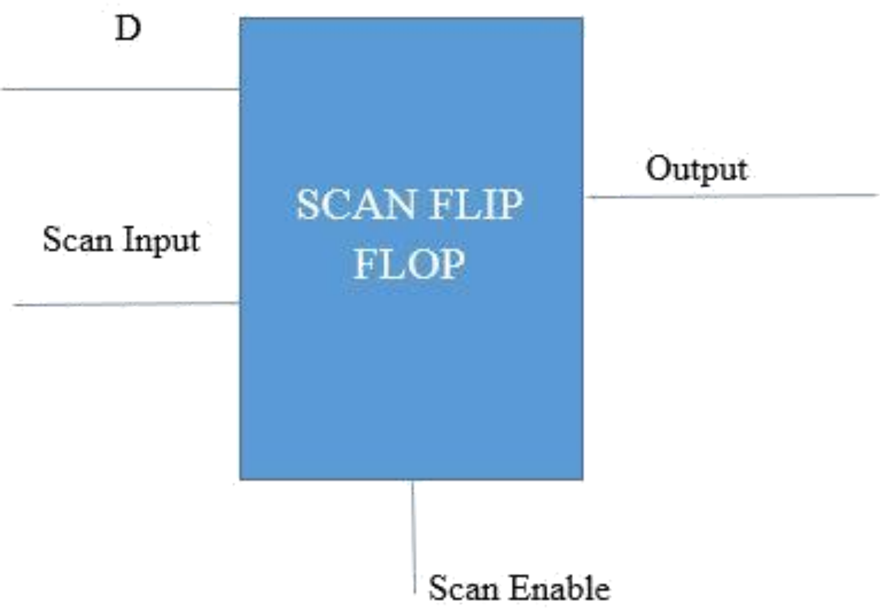


Figure 2: SCAN FLIP FLOP

iii) Scan Chains

Scan chains can be defined as the number of scan flip flops connected together serially. The figure shows the scan chain of length 'N' connected together. The scan chain length is defined as the number of scan cells in scan chain i.e. N.

The cell on the right hand side corner is numbered '0' and the then increases as we move towards left.

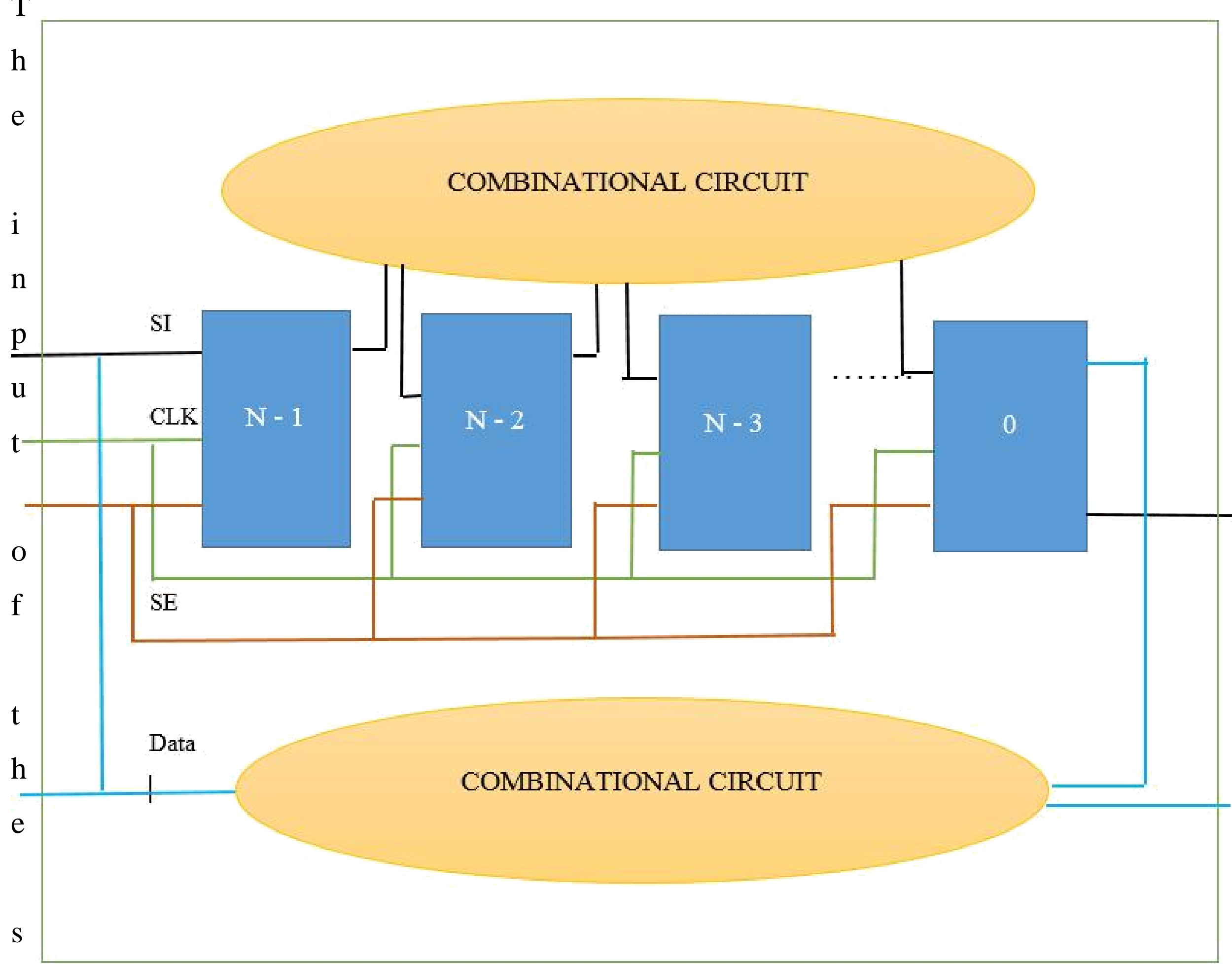


Figure 3 SCAIN CHAIN

The input of the scan chain is Scan-in and output is Scan-out. The least number of cells that can be present in a scan chain is 1. Here the clock given to all the scan chains is common. Also the scan enable used is common to all i.e. when the scan enable pin will be high the scan input will be activated. And also when clock is active the scan in will start to shift in the data.

When the number of scan chains is connected together in parallel and share a common test procedure file. The test procedure file will contain all the information by which scan chains of the group can be accessed.

If the two scan chains have a common scan-in input then they cannot operate in parallel. Scan groups are a way scan chains behave based on a operation. If the two scan chains have common scan – in signal it means they cannot operate parallel and that scan chain belongs to some other scan group.

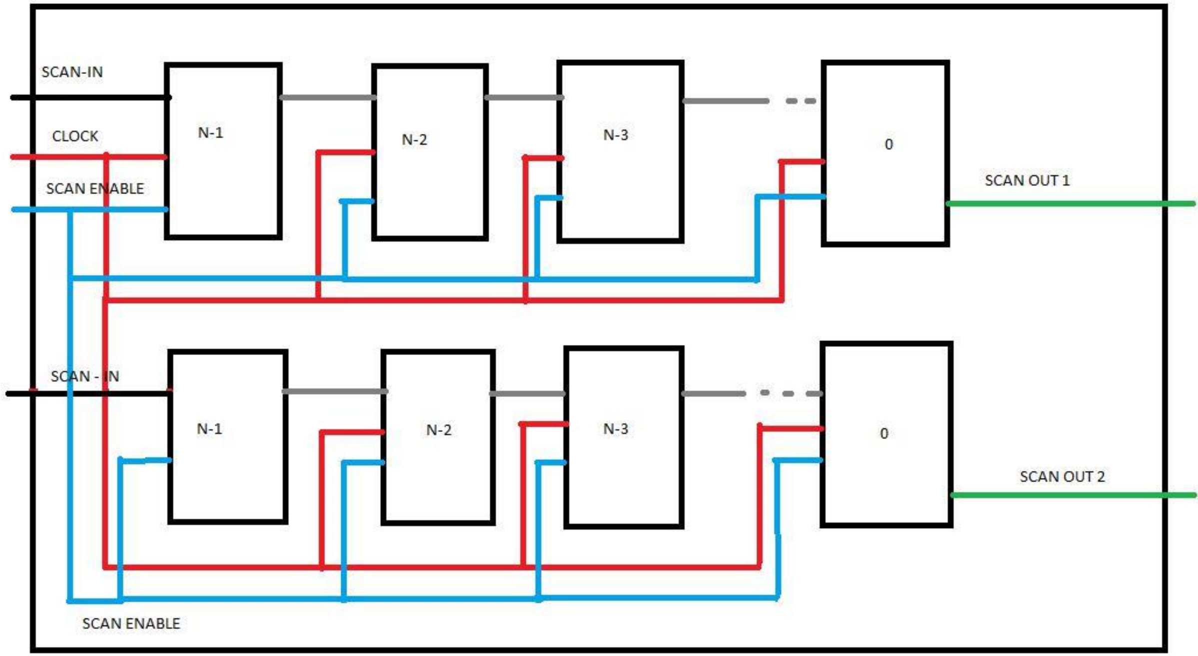


Figure 4 SCAN GROUP

iv) Scan Clocks

These are the external pins which can capture the data into the scan cells. When the scan – clocks are in ON position they capture the data and when in OFF position they hold/retain the data. When the scan clock is in OFF state it will deactivate the scan cell. When clear =1, then the flip flop goes in inactive state i.e. clock =0.

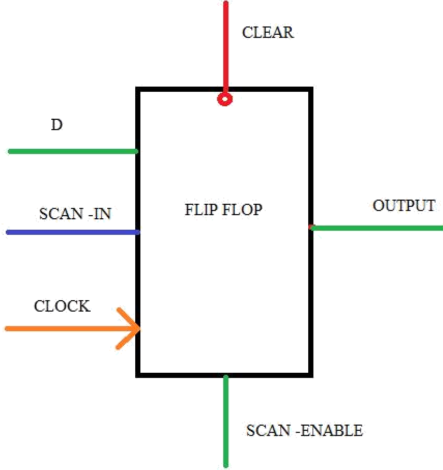


Figure 5 FLIP FLOP SHOWN WITH A CLOCK

(b) AUTOMATIC TEST PATTERN GENERATION (ATPG):

The basic goal of ATPG is to create a set of patterns for a design which when implemented on the design can give high test coverage. The patterns when applied to the design helps in differentiating the faulty and fault free circuit i.e. it determines the defects. The patterns also help us in determining the faults in the design. Now when the patterns are generated and then fed into the circuit, the number of faults determined or checked gives the fault coverage, test coverage and ATPG efficiency for the system.

FAULT REPORTING

i) Test Coverage:

It is defined as the ratio of the total number of faults detected to the total number of detectable faults.

$$\text{Test coverage} = \frac{\text{number of detected faults}}{\text{Total no. of detectable faults}}$$

ii) Fault Coverage:

It is defined as the ratio of total number of faults detected by the total number of faults present in the design.

$$\text{Fault coverage} = \frac{\text{Number of detected faults}}{\text{Total number of faults}}$$

Total number of faults = including all the faults which can be present in the design.

Total number of detectable faults = There are some faults in the design which are not detectable because of the design constraints and not because of the tool constraint. Such faults are difficult to detect as they are because of the limitation of our design.

Detectable faults = Total no. of faults – faults due to limitation of our design

Total number of faults > Number of detectable faults

So, Test coverage > Fault coverage

ATPG Effectiveness:

It measures the ATPG tool ability to either detect a fault and create a test pattern for it or prove the fault undetectable or untestable under the restrictions of the tool.

Fault Collapsing

A circuit may contain certain faults, some faults can be considered identical to each other. So, when the tool is working on identifying the faults it may not be able to distinguish

between the two faults. These kinds of faults are said to be equivalent to each other, and can be collapsed into one and this is called fault collapsing.

1.4. METHODS OF PATTERN GENERATION IN ATPG

i) Deterministic Pattern Generator:

In this we generate patterns according to the design requirement or as required in the particular fault. It will pick a fault, generate the pattern for that particular fault and then will simulate those patterns to make sure that fault is detected. In deterministic pattern generation we can find the redundant faults.

When we know the fault site, we will force it to the value opposite to the faulty value and would want to propagate it to the fault site. And then the particular pattern is selected which could debug that fault, this way the patterns generated are called deterministic patterns.

ii) Random Pattern Generator:

In this the ATPG will generate numerous patterns and then will detect the patterns which can detect faults. Then these patterns are stored in a separate file as a test pattern set. This kind of pattern generation is very different from the deterministic pattern generator as it can never identify the redundant faults and cannot create the patterns for low detection probability. It can be useful for the faults terminated by the deterministic pattern generator and can also increase the test coverage.

iii) External Test Pattern Generation:

This method is used when ATPG is done on the set of pre-existing values. The tool will analyze the pattern set and will execute the fault test from the fault list. It is an efficient method with high fault coverage.

1.5. TEST TYPES

i) Functional Tests:

It is the most widely used test. Here, we require the logical value of the input pins is required. It will apply the 1, 0 logic values at the input and will measure the output. So, the defect is proved when the input logic values differ from output logic values.

ii) IDDQ Tests:

It detects the failures not easily detected by the functional tests. For example: When making a CMOS Inverter, we check its efficiency which includes the leakage currents, quiescent current. When the power supply to the CMOS inverter is 0V, then ideally it should give zero current or pass zero current through it. But when without the power supply given to the circuit some value of current is still there, it can cause decrease in efficiency or power wastage. We use IDDQ tests to determine such faults.

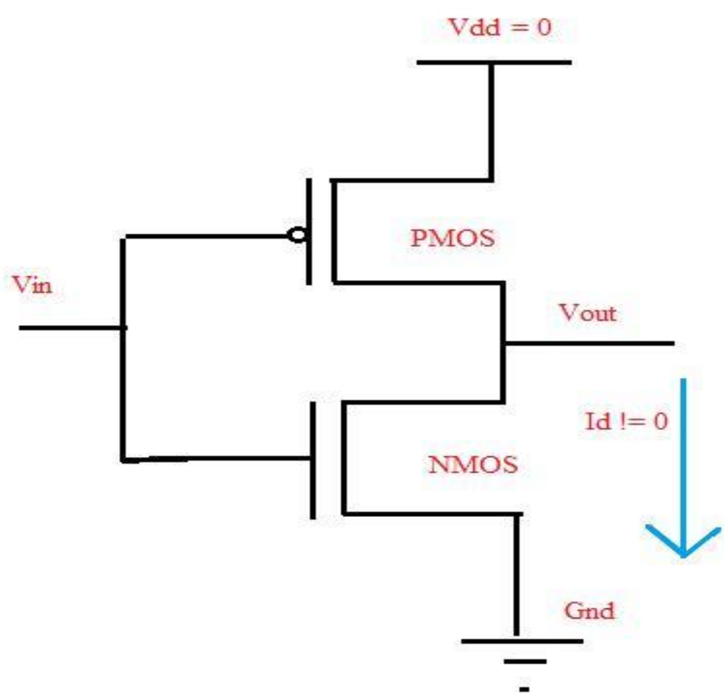


Figure 6 CMOS INVERTER WITH Vdd =0 AND NON ZERO CURRENT

1.5.1 IDDQ testing classified according to the CMOS circuits circuitry:

(a) Fully Static:

In Fully static CMOS circuits, the IDDQ current is near to 0. They have only one active driver. We can use any vector for the IDDQ current measurement.

(b) Resistive:

These have both pull-up and pull-down resistors which generate very high IDDQ current in a good circuit.

(c) Dynamic:

Dynamic CMOS circuits contain Macros (Library cells) that generate high IDDQ currents.

iii) AT-Speed Tests:

We generally test our circuit at low values in stuck- at faults. When the testing is done at low clock rate, and after that the circuit operates on its actual clock which is very high, then it fails during testing. So, we need At-Speed testing to ensure the proper working of the circuit at high clock rate. The one reason for which clocks are not tested on the application speed is the cost as testing on high clock rates becomes expensive.

1.6. ATPG APPROACH

Basically three steps are followed:

i) Fault Sensitization:

In this step we find out the point where we need to test the value, so we give such values at the input points such that the value at the input port is transferred to the fault site. The value transferred at fault site is always the opposite value.

ii) Fault propagation:

When we want to propagate the value to the fault site, I also need to set the values to the other input ports such that I am able to propagate the value to the fault site.

iii) Justification:

Finding the value at the input ports which will allow the neighboring inputs of fault site to be set to the value found in Fault Propagation step.

(c) SIMULATIONS

Simulations in DFT follow 3 steps:

i) COMPILATION:

When Netlist, Verilog files are compiled, the compilation checks the syntax of the code. If there is an error in the syntax it will be shown before proceeding further with the simulation steps. A tool called 'Parser' checks the files for syntax errors.

ii) ELABORATION:

This step comes after the compilation. If there is no syntax error we will go further to the elaboration check. In this the check for connected modules or connected instances or connected ports is there. If there is any port not connected properly or any connection from module is missing, it will generate the error. If all the connections are proper we can move to the next step.

iii) SIMULATION:

After successful compilation and elaboration, we will move to the simulation. After simulation, the design model is executed. DFT simulations are done with both timing and no-timing. Now, to check the setup firstly the files are run without delay or timing. Then we include the timing file afterwards which is called SDF (standard delay format) file.

SIMULATION BASED ON THE TIMING CONSTRAINTS:

i) No Time

In this type of simulation we do not consider any time constraints. Here the simulations are done considering zero delay, which means as soon as I give an input it will reach the output with zero delay. Here, we define the delay for sequential and combinational circuits differently but no other delay i.e. wires delay is considered. This is an ideal simulation for the design i.e. how it should behave in ideal conditions.

ii) Best Case

In this simulation, we will consider the SDF file which is given for the best case by the synthesis team. This file will contain all the parameters for the best case. The delay for combinational delay and sequential delay is also defined in it. These files are delivered to us by the synthesis team. Usually we do not find any timing constraint in this type of simulation.

iii) Worst Case

In this the SDF file given for Worst case is used while simulation. This file will contain all the parameters for worst case. The delay for the combinational and sequential circuits is defined. The meaning of the worst case means in the worst scenarios or conditions the chip is working fine i.e. at the highest temperature defined for it and its lowest temperature, etc. Suppose, our chip should work perfectly fine between 20 F and 80 F, so

in worst case scenario we will check the extreme conditions for our chip and will check the final result.

If some error is found in worst case, we match our timings with the best case results and check the outcomes.

So, we need all three conditions of our chip to be accurate for our chip to work fine.

CHAPTER -2

LITERATURE SURVEY

This section describes the work done by various researchers in the field of Design for Testability.

Steven D. Millman *et al.* [1] worked on different kinds of faults including bridging faults, transition faults and stuck-open faults. He presented a methodology for design that guarantees the checkers will be self-testing in the presence of bridging, stuck-at faults. Simulations confirm that these checkers are self-testing in the presence of bridging, stuck open and transition faults. Checkers are used for run-time error detections. In this paper, they have used the checkers in the CMOS design and proved with the simulations that checkers are able to detect all the stuck at and transition and bridging faults.

Keshava Iyengar Satish [2] describes the importance of JTAG technique, in his research such that as the design complexity is increasing we come across several issues while testing of chip. IC's offered by different manufacturers can interact with each other appropriately and also during testing of device mounted PCB is also important. So, this paper discusses about the several advantages of JTAG over the other testing techniques. The advantages of JTAG include increased controllability and observability, reuse of test set, better fault detection.

S. Barbagallo *et al.* [3] in his research describes that the ordering of the flip flops inside the scan chain should also be taken into account. It also describes two algorithms in the paper which takes into account the average and minimum distance between the flip flops and then reducing the power dissipation. The two algorithms are executed and their results prove the power dissipation in the circuit can be minimized by taking into account the number of scan cells in a chain, number of scan chains required also minimizing the fan outs.

Nikhil Sharma *et al.* [4] worked on a circuit such that the layout of a circuit can also decide the possibility of occurrence of faults. In this paper, he reduced the chances of occurrence of HTD (Hard to detect) faults. We define a fault *f* as an HTD fault if an automatic test pattern generator

fails to generate a test vector for f in a reasonable amount of CPU time. It is common practice to drop such HTD faults from consideration during test generation. The chip fault coverage achieved by a test set is poor if the fault set consists of many HTD faults. We can combat this problem by avoiding altogether, or by reducing the probability of, the occurrence of HTD faults. This paper describes module placement rules, with which we can reduce the probability of occurrence of HTD faults.

Hiroshi Yamazaki *et al.* [5] described the IDDQ testability for bridging faults in a variety of flip flops. A flip-flop with a deliberately introduced bridging fault is simulated by the SPICE simulator. Simulation results show that faults in some flip-flops cannot be detected by IDDQ testing, and this problem depends on the flip-flop structure. In this paper he discussed five different D-flip flop structures and with the help of simulations proved that the structure of Flip flop also decides the IDDQ testability.

L1. Ribas-Xirgo *et al.* [6] discusses that test generation for logic faults can be used to enable IDDQ sensing devices and enable a large number of IDDQ faults including stuck on and line bridging. But there are some fault models not covered by stuck on which require special attention. A symbolic simulation of a circuit having been injected a set of faults is performed to obtain its functional response, which is given as a set of functions in terms of input and fault-selection Boolean variables. Such functions are operated to obtain a minimal set of appropriate test vectors, which can be directly used as part of the final test set, or fed into a gate-level ATPG to improve the swatch-level fault coverage of its resulting test patterns.

Tsin-Yuan Chang *et al.* [7] in this work an IDDQ fault location scheme is proposed and implemented in a chip to locate single faulty branch (either a VDD branch or a GND branch or both) with three-pattern test by using hardware approach, which can be applied to the redundant VLSI structure for yield enhancement in the reparable design. The fault model includes IDDQ fault with or without single stuck on/ off fault in the added transistors.

Yoshinobu Higarni *et al.* [8] this paper presents a test generation method for sequential circuits assuming IDDQ testing. Here external bridging faults and internal bridging faults are considered as target fault. Test generation for external bridging faults consists of three phases as1) use of

weighted random vectors, 2) set of target values on selected signal lines, 3) deterministic test generation for undetected faults. In order to detect internal bridging faults, we use a sequential test generator for stuck-at faults. As the experimental results show, it insufficient for external bridging faults to use weighted random vectors and to set target values on stable lines. In order to obtain high coverage of internal bridging faults, it is necessary to generate test vectors which achieve high coverage of stuck-at faults.

Janak H. Patel [9] according to him, the stuck-at fault model cannot be directly considered related to physical defects. The main advantage of stuck-at fault is the abstraction which makes it worthwhile to be used in the millennial. This paper discusses two kinds of faults which can be considered in stuck-at i.e. Defects Internal to gate and Defects External to gate. The probability of detecting defects is discussed in this paper.

Wuudiann Ke *et al.* [10] he describes the design reuse environment in DFT. The test pins and vectors are accommodated in the chip from the beginning otherwise when an error is found late in the project it can lead to high cost and poor quality. So the test planning should be done in the beginning so that the DFT architecture is set accordingly and could deliver maximum efficiency in a project. This paper describes the various steps which should be executed while in DFT so that the maximum efficiency is achieved.

Eric Liau [11] described that the Automatic test equipment (ATE) used for the pattern testing is a slow process. This paper proposed an ATE for neutral networks which classified for worst case input patterns taking into account the maximum instantaneous currents. For further improvement, he developed an algorithm called genetic algorithm (GA).It detects the high switching currents which can cause increase in power supply leakage. This is the first neutral networks and genetic algorithm performed together.

C. Metra *et al.* [12] discussed about risks associated with DFT designs. As the chip size is decreasing and number of components on the chip is increasing so, the complexity is increasing which means that the testing is getting difficult. This paper discusses about the challenges faced during scan insertion, BIST, ATPG and then gives the better solution.

Seongmoon Wang *et al*. [13] he suggested that as there are some points in the scan flip flops which are untestable in the scan environment. So, we can insert the test points in the flip flops and combinational circuits which will be able to test those points. This method increases the fault coverage and also as it only inserts test points wherever necessary it decreases the area overhead. The researcher proved 40% decrease in the chain length by using the technique and also most of the transition delay faults were covered.

Irith Pomeranz *et al*. [14] described that the transition faults requires multiple clock cycles. As the transition faults considered multiple times, they gather extra delay on the faulty line. The model was introduced by the researcher which overcomes the short comings by introducing the values in the faulty circuit when the fault effects occur thus allowed all the faults occurring to be encompassed as the single fault. He introduced 3 versions of fault models with their corresponding fault coverage i.e. Pessimistic, Optimistic, and Random. This paper compares these three models along with their fault coverage.

Xiao-Xin FAN *et al.* [15] he described the faults in a system due to multiple clocks like: at-speed clock, synchronous clock. This paper presents the structure that uses a PLL (phase lock loop) as at-speed clock generator which will support the at-speed testing and will not increase the area overhead. Experimental results proved that the design has low area overhead even with the increase in number of clocks.

CHAPTER -3

PROBLEM STATEMENT

As discussed previously, we use embedded systems which include different software/hardware components to be used on the same chip. And according to the Moore's law the number of transistors on a chip doubles every two years. So, this shows that the VLSI industry is growing at a fast rate. The chip size is decreasing; the area constraint is not there. It means less area and better power management. But the only problem here is difficulty for testability.

As the complexity is increasing, the testability of chip is getting complex. If there are 1 million transistors on the chip, then I need to make a design such that all the components are tested well without affecting each other's functionality. It is often said that **Untested chip = Failed chip**. This statement gives the importance of DFT in the designing of the chip. DFT is such a field which is involved in the chip designing as soon as the RTL design is written i.e. Verilog code is written.

As shown in figure 7, the asic design flow shows different steps in the flow of making chip. The DFT come into power as soon as the Verilog code is delivered to them. The DFT will add their input pins like controllable clock pins or reset pins. The role here is quite small. Then the RTL is made and sent to synthesis team which will generate the netlist. Then the DFT team will replace the D flip flops with the scan able flip flops. This means scan insertion will be done. The scan flip flops will have a test enable pin, scan in pin. These flip flops are same as D flip flops but only some extra pins are added which will help us to check the data in the flops.

The scan flops are then connected together to form a scan chain. This scan chain has number of scan flip flops connected to each other. The purpose of forming a scan chain is to easily shift-in the data to the flops. We shift-in the data to the flops so that we can keep the flops in known state i.e. they are in control when we do the testability on them. After the scan insertion is done, i.e. forming scan chains and scan flops we go to the next step.

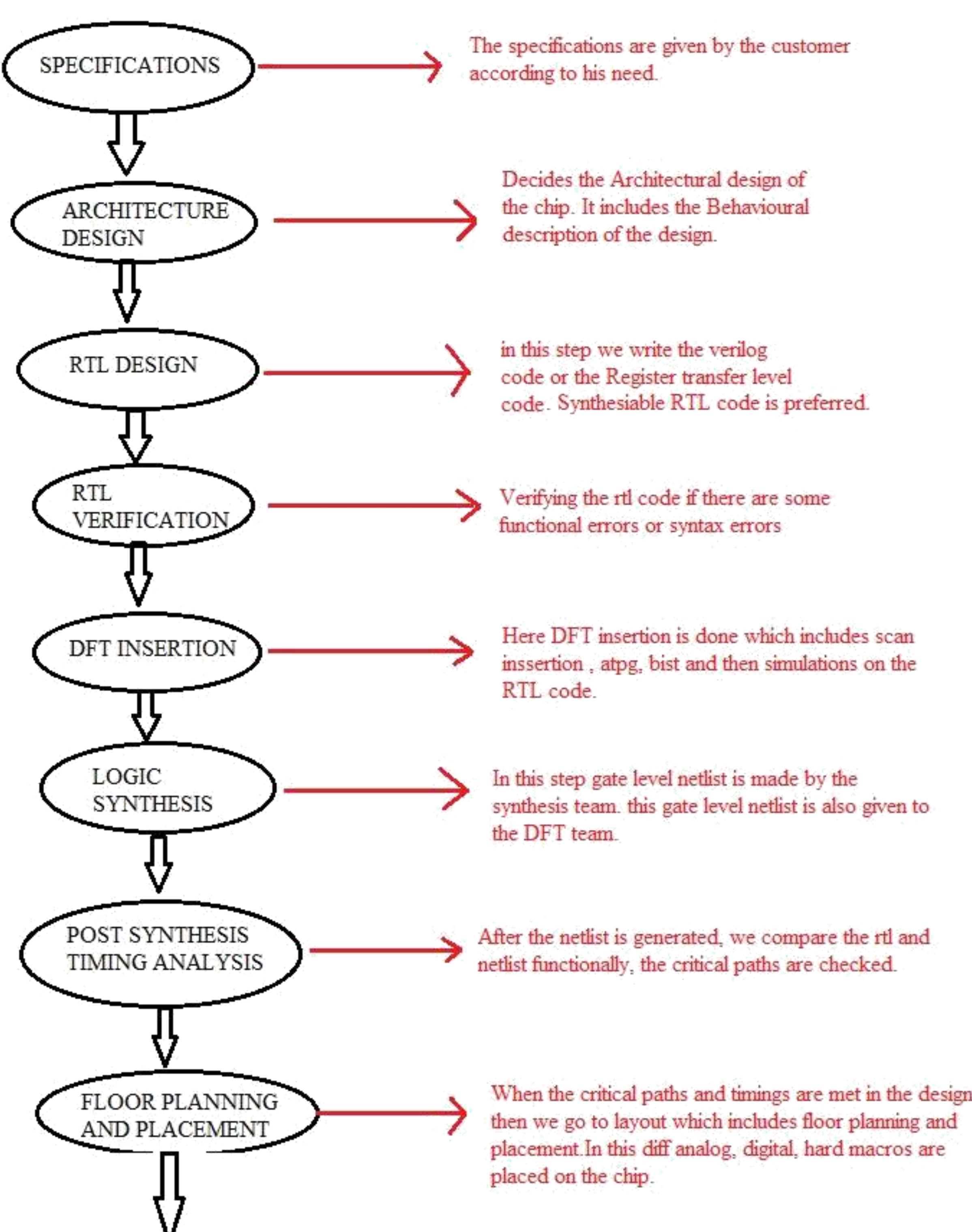
The next step is to do ATPG (Automatic test pattern generation), in this we generate different random patterns and then test these patterns on our scan chains and then do simulations to check the results. The simulations are done in different timing constraints i.e. worst case, best case and notime. After the Atpg is performed, we check the coverage of the design which is test coverage, in this we can check how many faults which are described in the design are covered. The coverage is different for different faults like: for stuck-at faults we require 99% coverage and for transition we require 90% minimum coverage. The more is the coverage of the faults the better is Atpg.

The basic requirement for industry is to deliver the good chips to the customer. The number of bad chips should be as low as possible. More is the number of good chips more will be the yield. So, our focus is to have high yield. To confirm high yield of good chips we need testability of the chip.

UNTESTED CHIP = FAULTY CHIP

This makes testability important. And thus, the role of DFT is an integral part in the asic design flow. The DFT is included in the 60 % - 70% of the chip designing.

Question*: As the complexity of the chip is increasing day by day, according to the Moore's law the number of components on a chip is doubled every 2 years. So one of the important issues to be considered is how to verify the chip in the convenient way such that it doesn't affect the other components on the chip. And also maximizes the test coverage. What methods are to be used in DFT which will maximize the test coverage and ensures maximum yield?*


SPECIFICATIONS
The specifications are given by the customer according to his need.
ARCHITECTURE DESIGN
Decides the Architectural design of the chip. It includes the Behavioural description of the design.
RTL DESIGN
in this step we write the verilog code or the Register transfer level code. Synthesiable RTL code is preferred.
RTL VERIFICATION
Verifying the rtl code if there are some functional errors or syntax errors
DFT INSERTION
Here DFT insertion is done which includes scan inssertion , atpg, bist and then simulations on the RTL code.
LOGIC SYNTHESIS
In this step gate level netlist is made by the synthesis team. this gate level netlist is also given to the DFT team.
POST SYNTHESIS TIMING ANALYSIS
After the netlist is generated, we compare the rtl and netlist functionally, the critical paths are checked.
FLOOR PLANNING AND PLACEMENT
When the critical paths and timings are met in the design then we go to layout which includes floor planning and placement.In this diff analog, digital, hard macros are placed on the chip.

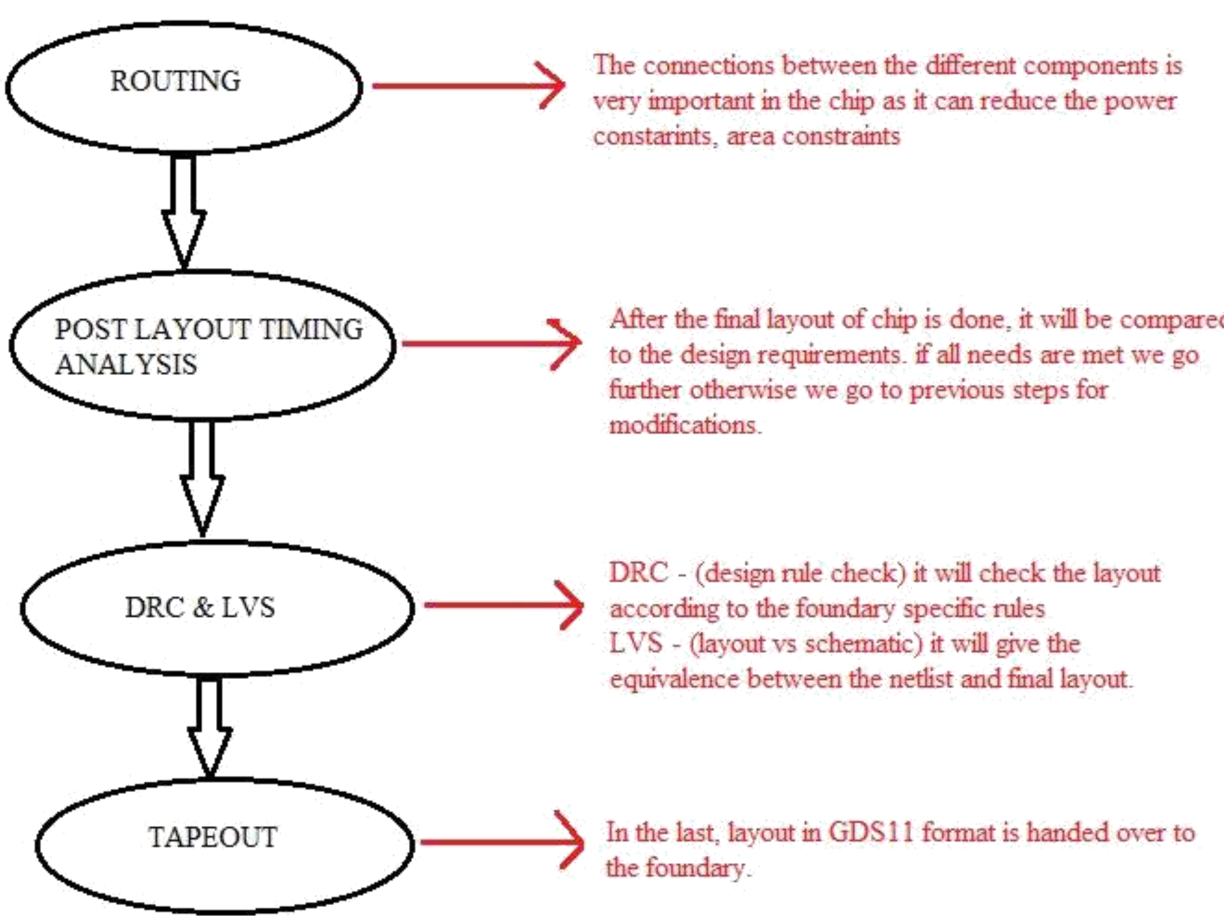


Figure 7ASIC DESIGN FLOW

CHAPTER – 4

WORK DONE AND SIMULATIONS

PROJECT NAME - Design for Testability Pattern Generation and Simulation of Complex Circuits.

Working Environment - UNIX

As discussed previously we follow 3 main steps in DFT to ensure proper testability of the chip.

i) SCAN INSERTION

STEP 1: Firstly we set the context to dft. setting context means telling your tool about the current usage. So, when we write command *set_context dft –scan,* it will set the current usage to dft and – scan means we are going to do scan insertion.

STEP 2: Reads one or more Verilog files into the specified or default logical library. The command is *read_verilog.../netlists/cpu.v, here* cpu.v file will be read by the tool.

STEP 3: Read the cell libraries into the tool.
read_cell_library.../lib/ADK_4.2/tessent/adk.tcelllib, this command will load cell libraries into the tool.

STEP 4: Now the tool will specify the top level design. *set_current_design*, this command when issued without an argument will find the top level of the design if it finds more than one top level, it will present the list of all possible top levels and then we need to run the command again with the specific top level.

STEP 5: Next step is to add clocks in the design. The command used for it is *add_clock.* This will add synchronous clock into the design. We can give a default value to the clock in the

beginning like: *add_clock 0 clock1*, this will add a clock named clock 1 with a default value of 0, it means the clock 1 is in off state when given value 1.

NOTE: For ATPG and SCAN DRC only synchronous clocks are supported.

STEP 6: Next step is to add cell model. *add_cell_models latch -type dlat CLK D*

Model is the keyword that describes a single library cell in the technology library. When adding test logic circuitry, multiple models from the DFT library are used.

The *cell_type* attribute in the library model description specifies the components available for use as test logic.

-type will specify the model type. In this case, the model type is D latch.

D latch has 2 pins i.e. enable and data so here enable is CLK and data is D.

STEP 7: After adding cell_models next we want is to check if we are violating any design rules.

The command used is *check_design_rules*. Basically, our system works in 2 modes: one is the setup mode and other is the analysis mode.

Setup Mode: In this mode, we can s*et al.*l the data into the design like: adding clocks, adding cell models, setting the environment to dft. After all the data is fed to the tool for the design we can move to the next mode which is analysis mode.

Analysis Mode: In this mode, we have to check the data added in during the setup mode which is checking the design rules if any is violated.

So, in this step we need to check the design rules. We check the DRC rules specific to the context. If no DRC failure is there, we enter the analysis mode. If any DRC failure occur the tool will remain in the setup mode which enables to analyze DRC violation.

STEP 8: In this step test logic is inserted. Command is *insert_test_logic -number 4 -clock merge -edge merge*. It will insert the test structure defined in the netlist and will stitch up scan chains to increase the testability of the circuit.

-number 4: The number of scan chains you want the tool to insert. Here the number = 4, it means 4 scan chains. By default, it inserts 1 scan chain.

- *clock merge:* It will specify that whether a single clock or different clocks will be there for different scan chains. 'merge' specifies that different clocks should be used for scan chains. 'unmerge' specifies single clock for all chains.

-edge merge: It gives the option whether the tool should merge stable high chains into stable low chains. 'merge' keyword specifies to merge the stable high chains with stable low chains. 'unmerge' specifies not to merge the stable high chains to stable low chains.

STEP 9: In this step we want the description of all the scan chains added in a file format. So, the next command is *report_scan_chains.* This command will give the detailed report of:

- Name of scan chain
- Name of scan chain group
- Scan chain input and output pins
- Length of the scan chain
- Name of the scan clock(s)

STEP 10: This step will give the information about all the scan cells present in the scan chains. We use the command *report_scan_cells, it* will give the detailed report of:

- Will display the cell ID or index number. Here, cell with 0 index number is closest to the scan out pin.
- It will give the detail of the scan chain in which the scan cell resides.
- Details of the scan group where the scan is there. If there is only one scan group then it will not give any result.
- Information of the sequential element i.e. it's type.
- Library model name for all the scan cells.
- Instance names associated with each sequential elements.
- Scan cell input and output pins.
- If we want the information about a specific scan cell then I will give its path name with the command, otherwise it will generate the report for all the scan cells.

STEP 11: Next step involves writing the current design in Verilog format in the specified file. The command used is *write_design -output_file .../netlists/cpu_aman_scan.v –replace.*

Here the –output file means we need to write the design in the output file which is named top_scan.v

-replace written in the end indicates that whatever maybe written in the output file previously, we have the right to replace it.

STEP 12: In this step we write a test procedure file and do file for the existing scan chains. Using command *write_atpg_setup.../2_atpg/atpg_setup_aman–replace*. You can use these files to identify the scan chains when loading the scan design into the tool.

Using –replace confirms that the contents of the file will be replaced.

After doing these steps, a log file will be generated with the name as provided in the write_design and write_atpg_steup command.

TOOL USED – Tessent Shell, version 2016.2 by Mentor Graphics

```
//
//  command: ##
//  command: ## Scan insertion flow
//  command: ##
//  command: set_context dft -scan
//  command: read_verilog ../netlists/cpu.v
//  command: read_cell_library ../lib/ADK_4.2/tessent/adk.tcelllib
//  Reading DFT Library file ../lib/ADK_4.2/tessent/adk.tcelllib
//  Finished reading file ../lib/ADK_4.2/tessent/adk.tcelllib
//  command: set_current_design
//  Note: Top design is 'cpu'.
//  command: add_clocks 0 CLK1
//  command: add_clocks 0 CLK2
//  command: add_clocks 0 CLK3
//  command: add_clocks 0 RESET
//  command: add_clocks 0 TRESET
//  command: add_cell_models latch -type dlat CLK D
//  command: check_design_rules
//  Warning: Rule FN1 violation occurs 3 times
//  Warning: Rule FN4 violation occurs 434 times
//  Flattening process completed, cell instances=4835, gates=7723, PIs=79, POs=148, CPU time=0.03 sec.
//  -------------------------------------------------------------------------
//  Begin circuit learning analyses.
//  --------------------------------
//  Learning completed, CPU time=0.02 sec.
//  -------------------------------------------------------------------------
//  Begin scan chain identification process, memory elements = 434.
//  -------------------------------------------------------------------------
//  -------------------------------------------------------------------------
//  Begin scannable cell rules checking for 434 nonscan memory elements.
//  -------------------------------------------------------------------------
//  431 non-scan memory elements identified as scannable.
//  -------------------------------------------------------------------------
//  Begin scan clock rules checking.
//  -------------------------------------------------------------------------
//  5 scan clock/set/reset lines have been identified.
//  All scan clocks successfully passed off-state check.
//  431 sequential cells passed clock stability checking.
//  -------------------------------------------------------------------------
//  3 non-scan memory elements are identified.
//  -------------------------------------------------------------------------
//  3 non-scan memory elements are identified as INIT-X. (D5)
//  -------------------------------------------------------------------------
//  Warning: There were 3 S1 violations (unstable nonscan cells when clocks off).
//  -------------------------------------------------------------------------
                                                                                 54,49          2%
```

Figure 8 LOG FILE GENERATED AFTER SCAN INSERTION PROCESS

```
  scan_enable = /scan_en  clock = /CLK2
chain = chain2   group = dummy  input = /scan_in2  output = /scan_out2  length = 108
  scan_enable = /scan_en  clock = /CLK2 /CLK1
  reset = /TRESET /RESET
chain = chain3   group = dummy  input = /scan_in3  output = /scan_out3  length = 108
  scan_enable = /scan_en  clock = /CLK2
chain = chain4   group = dummy  input = /scan_in4  output = /scan_out4  length = 107
  scan_enable = /scan_en  clock = /CLK3 /CLK2
//  command: report_scan_cells
------------------------------------------------------------------------------------------------
CellNo  ChainName  GroupName  Pathname                        CellName  ScanOut  Clock  ClockPolarity
------------------------------------------------------------------------------------------------
0       chain1     dummy      /uDMI/ix450                     sff       Q        CLK2   (+)
1       chain1     dummy      /uDMI/ix500                     sff       Q        CLK2   (+)
2       chain1     dummy      /uDMI/ix470                     sff       QB       CLK2   (+)
3       chain1     dummy      /uDMI/ix420                     sff       Q        CLK2   (+)
4       chain1     dummy      /uDMI/ix330                     sff       QB       CLK2   (+)
5       chain1     dummy      /uDMI/ix510                     sff       Q        CLK2   (+)
6       chain1     dummy      /uDMI/ix490                     sff       QB       CLK2   (+)
7       chain1     dummy      /uDMI/ix520                     sff       QB       CLK2   (+)
8       chain1     dummy      /uDMI/ix530                     sff       QB       CLK2   (+)
9       chain1     dummy      /uDMI/ix540                     sff       QB       CLK2   (+)
10      chain1     dummy      /uDMI/ix550                     sff       QB       CLK2   (+)
11      chain1     dummy      /uDMI/ix280                     sff       QB       CLK2   (+)
12      chain1     dummy      /uDMI/ix260                     sff       QB       CLK2   (+)
13      chain1     dummy      /uDMI/ix240                     sff       QB       CLK2   (+)
14      chain1     dummy      /uDMI/ix230                     sff       QB       CLK2   (+)
15      chain1     dummy      /uDMI/ix250                     sff       QB       CLK2   (+)
16      chain1     dummy      /uDMI/ix270                     sff       QB       CLK2   (+)
17      chain1     dummy      /uDMI/ix310                     sff       QB       CLK2   (+)
18      chain1     dummy      /uDMI/ix320                     sff       QB       CLK2   (+)
19      chain1     dummy      /uDMI/ix340                     sff       Q        CLK2   (+)
20      chain1     dummy      /uDMI/ix300                     sff       Q        CLK2   (+)
21      chain1     dummy      /uDMI/ix290                     sff       Q        CLK2   (+)
22      chain1     dummy      /uDMI/ix360                     sff       Q        CLK2   (+)
23      chain1     dummy      /uDMI/ix350                     sff       Q        CLK2   (+)
24      chain1     dummy      /uDMI/ix380                     sff       Q        CLK2   (+)
25      chain1     dummy      /uDMI/ix370                     sff       Q        CLK2   (+)
26      chain1     dummy      /uDMI/ix390                     sff       Q        CLK2   (+)
27      chain1     dummy      /uDMI/ix480                     sff       Q        CLK2   (+)
28      chain1     dummy      /uDMI/ix410                     sff       Q        CLK2   (+)
29      chain1     dummy      /uDMI/ix400                     sff       Q        CLK2   (+)
30      chain1     dummy      /uDMI/ix440                     sff       Q        CLK2   (+)
31      chain1     dummy      /uDMI/ix430                     sff       Q        CLK2   (+)
32      chain1     dummy      /uDMI/ix460                     sff       Q        CLK2   (+)
                                                                                    74,39          15%
```

Figure 9 LOG FILE GENERATED AFTER SCAN INSERTION PROCESS

ii) AUTOMATIC TEST PATTERN GENERATION

After doing scan insertion, the major steps involved in it are:

- We changed the flip flops to scan flip flops
- Added clocks in the design
- Added cell models
- Added necessary libraries
- Checked design rules
- Added scan chains, decided the length of the scan chains

In scan insertion, 4 files are generated after the scan insertion process is over:

- LOG File is generated.
- Do File
- Test Procedure File
- Netlist after scan insertion is done

Now we will move onto the next step which is ATPG. In this step we will generate patterns which will test the existing design.

STEPS INVOVLVED IN ATPG:

We have got two input files from the previous procedure which are Test procedure file, ATPG do file.

STEP 1: The first is to read the "do" file in ATPG setup.

STEP 2: The first step is to read the Netlist generated in the scan insertion process, which includes DFT designs.

STEP 3: In next step we include and read all the libraries associated with the design.

STEP 4: In this step command used is *set_current_design.*This command will specify the top level of the design. When more than one top level exists, the tool will specify the list and you can choose one.

STEP 5: In next step we refer to the do file generated after scan insertion process,according to the do file given, first step is to add clocks in the design. Using the command '*add_clock default value pin name'*

STEP 6: The next step involves adding the scan groups, which are defined in the test procedure file. So, now entering the test procedure file.

STEP 7: In this step we enter the test procedure file. The various steps involved in test procedure file are:

i) Firstly, the Time scale is given. If no time scale is given it is taken by default equal to 1 ns.

ii) Next step defines the Time plate. Time plate name is given in the beginning.

What is a Time Plate?

Time plate defines the single tester cycle and specifies all the events placed in cycle. All clocks must be defined in the time plate definition.

Timeplate gen_aman =

iii) Now we will force the values which are there in this time plate.

'force_input 0' – In this step the input values are forced related to a particular time plate. Here 0 gives the time at which these values are forced in the input.

'measure_output 10' - Here we measure the output value after the input value is forced, after 10ns. It means the output is checked after a certain strobe window.

*'pulse_clock 20 10' – pulse*_clock time width

It is an integer that specifies the pulse timing for all signals defined as clock until another force or pulse exists for a particular clock signal.

We use pulse_clock to ensure that any added clocks (including internal clocks) have automatically defined timing.

'Pulse_clock' is the keyword

'Time' defines the offset from time 0 to the leading edge of the pulse.

'Width' defines width of the pulse.

Pulse clk1 20 10

Pulse reset 20 10 – Here pulse is defined in which rest of the clock signals can be defined. Here again, time and width are defined for a particular signal.

Period 40 – This is the last step in the definition of time plate. We define the period of tester cycle. It ensures that the cycle contains sufficient time after the last force event for the circuit to stabilize.

The time specifies should be greater than or equal to final event time.

iv) After giving the template definition, we move onto the next step. The next step involves defining the procedure to shift the values into the scan chains.
We also define scan groups here, the scan groups are made with several scan chains which are to be given the same value during the shift procedure.
procedure shift
scan_group grp1
timeplate gen_aman
cycle =
force_scaninput;
measure_scanoutput;
pulse clk1;

pulse clk2;

end;

end;

In the procedure of adding values to the scan chains, we define it for different groups in the tool. Then again forcing the input and measuring the output after a strobe window.

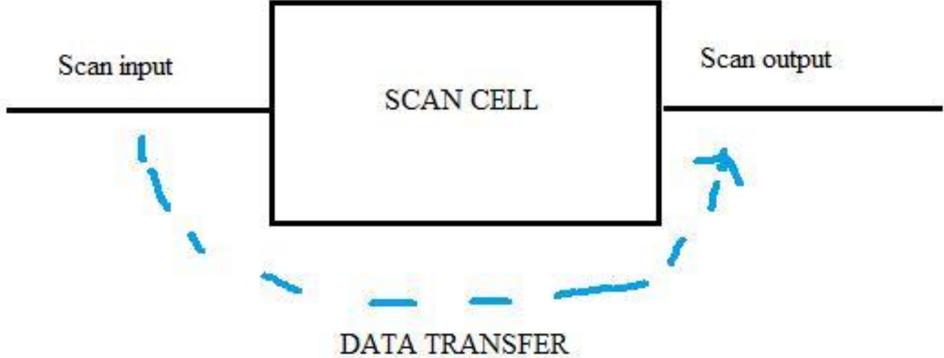


Figure 10 SHOWS HOW THE DATA IS TRANSFERRED IN PROCEDURE SHIFT

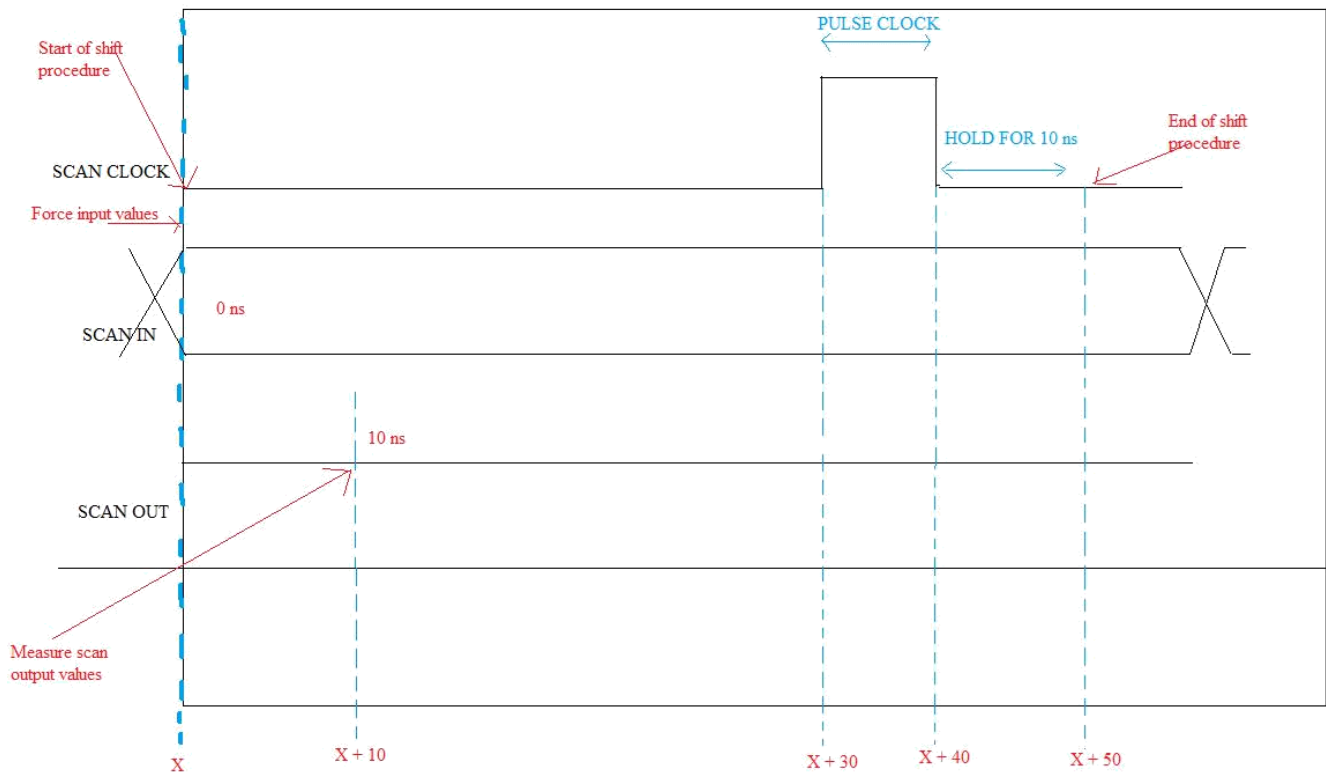


Figure 11 PROCEDURE SHIFT PROCESS

From figure 11 we get certain conclusions about the shifting procedure:

- Force scan input values at 0 ns.
- Measures scan output values at 10 ns.
- Pulse the scan clock [Turning it on at 30 ns and turning off at 40 ns].
- Then holding the state of last event until procedure is finish at 50 ns.

v) After doing the shift procedure, the next step is to do the loading and unloading procedure.

procedure load_unload

= scan_group grp1 ;

```
timeplate gen_aman ;
cycle =
   force CLK1 0 ;
  force RESET 0 ;
  force TRESET 0 ;
  force scan_en 1 ;
end;
 apply shift 216;
end;
```

Procedure load unload describes how to load and unload the scan chains in the scan group. To load the scan chains, initially force it to an appropriate state for the start of the shift sequence. This includes forcing the clocks, resets or other signals which are required to be in off state for the loading of the chain.

Apply shift– this is the command which will specify the number of shift cycles. Also the shift cycles are equal to the scan elements in a scan chain.

Also if there is a scan enable signal present, it should be force to 1 so that scan chain shifting is enabled.

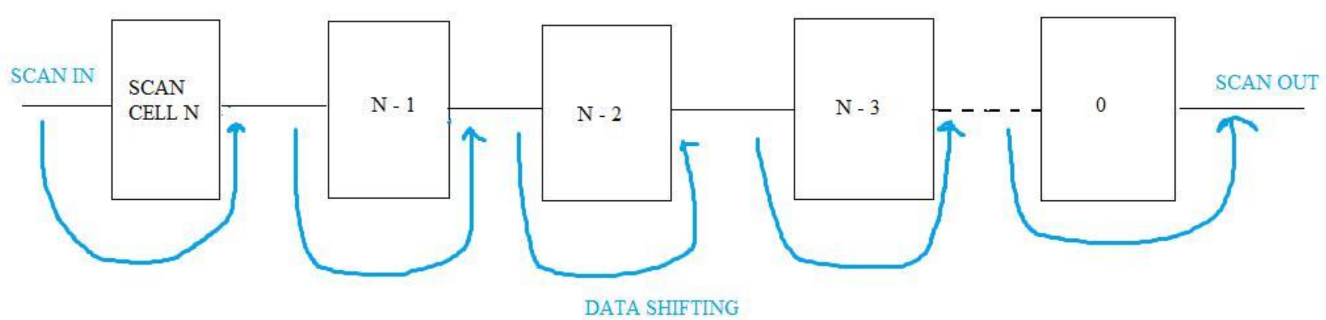


Figure 12 DATA SHIFTING IN LOADING AND UNLOADING PROCEDURE

After data loading and unloading we exit the procedure file and move to the Atpg Do file again.

STEP 8: Next step is to add the existing scan chains. Command used is

add_scan_chain chain1 grp1 scan_in1 scan_out1

add_scan_chains chain2 grp1 scan_in2 scan_out2

add_scan_chain chain name group name input pin output pin'

It will specify the name for the pre-existing scan chains in the design. A pre-existing scan chain is the one which is stitched using various scan cells in the previous scan operation.

STEP 9: Till this step the Atpg do file and Atpg procedure file is complete. Now we want to enter the analysis mode, which will check if there is any violation involved till this point.

Using command *set_system_mode analysis.* It will change the mode from setup to analysis.

The default mode for set analysis is the setup mode.

STEP 10: The next command is given *create_patterns.*

In this step, mode is analysis mode. It will perform automatic test generation and pattern compression. It will produce highly efficient and high quality test patterns.

The create_patterns command displays the following statistics:

- Number of patterns simulated
- Test coverage
- Number of faults detected
- Number of faults remaining in the fault list (undetected)
- Number of effective patterns
- Number of test patterns
- Number of redundant, untestable, and aborted faults

STEP 11: In next step we use command *report_statistics.* The report statistics command will display the complete report on the screen. It also displays the CPU processing time.

The report_statistics command displays the following statistics:

- The fault classes and percentage of different faults in the design.
- The total coverage in the design including test coverage, fault coverage and ATPG effectiveness.

STEP 12: In the last step we want to save the patterns generated in required format in the file. We use command

write_patterns -verilog -parallel -replace

write_patterns -verilog -begin 0 -end 2 -serial –replace

It will save the current test pattern set to a file in a specified format. Here, we are generating both serial and parallel patterns. In serial patterns we have given both beginning and end pattern numbers. After this the respective files are saved at their destinations.

Now, after pattern generation is done we go to simulations.

TOOL USED – Tessent Shell, version 2016.2 by Mentor Graphics.

```
//  command: ## Set Context, read scan inserted netlist and atpg library files
//  command: ##
//  command: set_context patterns -scan
//  command: read_verilog ../netlists/cpu_scan.v
//  command: read_cell_library ../lib/ADK_4.2/tessent/adk.tcelllib
//  Reading DFT Library file ../lib/ADK_4.2/tessent/adk.tcelllib
//  Finished reading file ../lib/ADK_4.2/tessent/adk.tcelllib
//  command: set_current_design
//  Note: Top design is 'cpu'.
//  command: ##
//  command: ## Reading atpg setup generated by during scan insertion
//  command: ##
//  command: dofile atpg_setup.dofile
//    command: add_clocks 0 CLK3
//    command: add_clocks 0 TRESET
//    command: add_clocks 0 RESET
//    command: add_clocks 0 CLK1
//    command: add_clocks 0 CLK2
//    command: add_scan_groups grp1 ../2_atpg/atpg_setup.testproc
//    command: add_scan_chains chain1 grp1 scan_in1 scan_out1
//    command: add_scan_chains chain2 grp1 scan_in2 scan_out2
//    command: add_scan_chains chain3 grp1 scan_in3 scan_out3
//    command: add_scan_chains chain4 grp1 scan_in4 scan_out4
//  command: ##
//  command: ## Set system mode to analysis - Runs design rule check
//  command: ##
//  command: set_system_mode analysis
//  Warning: Rule FN1 violation occurs 4 times
//  Warning: Rule FN4 violation occurs 436 times
//  Flattening process completed, cell instances=4839, gates=8196, PIs=84, POs=152, CPU time=0.03 sec.
//  ---------------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ------------------------------
//  Learning completed, CPU time=0.02 sec.
//  ---------------------------------------------------------------------------
                                                                          44,1          4%
```

Figure 13LOG FILE AFTER ATPG [1]

```
              Statistics Report
               Stuck-at Faults
--------------------------------------------
Fault Classes                       #faults
                                    (total)
----------------------------  -------------
  FU (full)                          33840
  --------------------------  -------------
  UC (uncontrolled)               998 ( 2.95%)
  UO (unobserved)                   7 ( 0.02%)
  DS (det_simulation)           18757 (55.43%)
  DI (det_implication)           3060 ( 9.04%)
  UU (unused)                     200 ( 0.59%)
  RE (redundant)                  364 ( 1.08%)
  AU (atpg_untestable)          10454 (30.89%)
--------------------------------------------
Fault Sub-classes
  --------------------------
  AU (atpg_untestable)
    SEQ  (sequential_depth)        12 ( 0.04%)
    Unclassified                10442 (30.86%)
  UC+UO
    AAB  (atpg_abort)            1003 ( 2.96%)
    Unclassified                    2 ( 0.01%)
--------------------------------------------
Coverage
  --------------------------
  test_coverage                         65.56%
  fault_coverage                        64.47%
  atpg_effectiveness                    97.03%
--------------------------------------------
#test_patterns                             294
  #basic_patterns                          274
  #clock_sequential_patterns                20
#simulated_patterns                        325
CPU_time (secs)                           10.3
--------------------------------------------

//  command: ##
//  command: ## report the test coverage
//  command: ##
//  command: report_statistics
                                                                          172,1         78%
```

Figure 14 LOG FILE AFTER ATPG PROCESS [2]

```
  FU (full)                        33840
  -------------------------  ------------
  UC (uncontrolled)              998 ( 2.95%)
  UO (unobserved)                  7 ( 0.02%)
  DS (det_simulation)          18757 (55.43%)
  DI (det_implication)          3060 ( 9.04%)
  UU (unused)                    200 ( 0.59%)
  RE (redundant)                 364 ( 1.08%)
  AU (atpg_untestable)         10454 (30.89%)
--------------------------------------------
Fault Sub-classes
  -------------------------
  AU (atpg_untestable)
    SEQ  (sequential_depth)       12 ( 0.04%)
    Unclassified               10442 (30.86%)
  UC+UO
    AAB  (atpg_abort)           1003 ( 2.96%)
    Unclassified                   2 ( 0.01%)
--------------------------------------------
Coverage
  -------------------------
  test_coverage                       65.56%
  fault_coverage                      64.47%
  atpg_effectiveness                  97.03%
--------------------------------------------
#test_patterns                           294
  #basic_patterns                        274
  #clock_sequential_patterns              20
#simulated_patterns                      325
CPU_time (secs)                         10.3
--------------------------------------------

//  command: ##
//  command: ## Save parallel and serial verilog patterns for simulation
//  command: ##
//  command: write_patterns ../3_vsim/pat_stuck_par_aman.v -verilog -parallel -replace
//  command: write_patterns ../3_vsim/pat_stuck_ser_aman.v -verilog -begin 0 -end 2 -serial -replace
//  command: ##
//  command: ## exit
//  command: ##

                                                                                                                       220,1         Bot
```

Figure 15 LOG FILE AFTER ATPG PROCESS [3]

iii) SIMULATIONS

This is the last step involved. In this step, we will check the correctness of the design by simulating patterns on the design. The pattern files required are generated from the previous step i.e. ATPG.

After successfully completing ATPG we get below files:

- Verilog file containing all the patterns for both serial and parallel patterns.
- File containing the chain names for both serial and parallel patterns.
- Configuration files for both serial and parallel patterns.

Now to simulate these files we will go to the .do file for both serial and parallel patterns, which will guide us further about the steps to be followed.

Simulations basically undergo 3 main steps i.e. Compilation, Elaboration and Simulation

STEP 1: We make a separate directory in which simulation log files will be stored. We need the library files, test bench, netlist to simulate.

STEP 2: The first step is to compile the .v file which is the netlist given. We will use the ncsim to compile the test bench.

ncvlog -APPEND_LOG -LOGFILE ncvlog_log_aman.log -DEFINE functional -HDLVAR hdl.var -work board_lib 'path for library file is given'

ncvlog -APPEND_LOG -LOGFILE ncvlog_log_aman.log -DEFINE functional -HDLVAR hdl.var -work board_lib ./'name of the testbench'

Hdl File = this file contains the paths of different directories which are required to be included while simulation.

Cds File = this file has all the libraries included.

In the above two commands we compiled the test bench and library files successfully. While doing compilation, the tool checks for any syntax command error or if something is undefined or if improper naming is done or improper assignment is done. All this is checked under compilation, after compilation is done, we move to Elaboration.

```
        module board_lib.iopad
                errors: 0, warnings: 0
        module board_lib.ipad
                errors: 0, warnings: 0
        module board_lib.opad
                errors: 0, warnings: 0
        module board_lib.opad_and
                errors: 0, warnings: 0
        module board_lib.ipad_buf
                errors: 0, warnings: 0
        module board_lib.sync_cell
                errors: 0, warnings: 0
        module board_lib.clock_buf02
                errors: 0, warnings: 0
        module board_lib.clock_inv01
                errors: 0, warnings: 0
        module board_lib.clock_mux21
                errors: 0, warnings: 0
        module board_lib.clock_and02
                errors: 0, warnings: 0
        module board_lib.clock_or02
                errors: 0, warnings: 0
TOOL:   ncvlog(64)      15.20-s010: Exiting on May 28, 2019 at 12:11:09 IST  (total: 00:00:01)

---

ncvlog(64): 15.20-s010: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
```

Figure 16 LOG FILE GENERATED AFTER COMPILATION

STEP 3: After all the files are compiled we will do elaboration for the test bench.

ncelab -APPEND_LOG -LOGFILE $COMPLOG -messages -work board_lib board_lib. 'name of test bench'

In this step, we did the elaboration. In elaboration the tool checks the proper connections between the different ports or if some port or something is left unconnected or if proper library files are included in the design while elaboration. If anything is missing we get an error. Here, we have elaborated successfully.

```
module board_lib.block16_scan
        errors: 0, warnings: 0
module board_lib.block17
        errors: 0, warnings: 0
module board_lib.block15
        errors: 0, warnings: 0
module board_lib.block14
        errors: 0, warnings: 0
module board_lib.block13
        errors: 0, warnings: 0
module board_lib.block12
        errors: 0, warnings: 0
module board_lib.block10
        errors: 0, warnings: 0
module board_lib.block02u
        errors: 0, warnings: 0
module board_lib.block11
        errors: 0, warnings: 0
module board_lib.block09
        errors: 0, warnings: 0
module board_lib.block08
        errors: 0, warnings: 0
module board_lib.block07
        errors: 0, warnings: 0
module board_lib.block06
        errors: 0, warnings: 0
module board_lib.block04
        errors: 0, warnings: 0
module board_lib.block05
        errors: 0, warnings: 0
module board_lib.block02
        errors: 0, warnings: 0
module board_lib.block01
        errors: 0, warnings: 0
module board_lib.block03
        errors: 0, warnings: 0
module board_lib.cpu
        errors: 0, warnings: 0
```

Figure 17 ELABORATION COMPLETED WITH ZERO ERRORS

STEP 4: After both elaboration and compilation are done, next step is to finally simulate the design. In simulation we check the expected results with the actual results we are getting, if there is a mismatch we have to debug that further.

```
Running simulation with license features associated with mnemonic 'IESXL'...
        Incisive_Enterprise_Simulator 15.20, 1 copy - Successful
Loading snapshot board_lib.cpu_pat_stuck_v_ctl:module .................... Done
ncsim> source /sw/cadence/incisive/14.20.010/lnx86/tools/inca/files/ncsimrc
ncsim> database -open -shm -into ./ncsim_shm.shm  -event  -default
ncsim: *E,DBMSNM: Missing database logical name.
ncsim> probe -create -shm cpu_pat_stuck_v_ctl  -depth all -all
Created default SHM database ncsim.shm
Created probe 1
ncsim>
ncsim> run
Loading pat_stuck.v.0.vec

Begin chain test

End chain test

No error between simulated and expected patterns

Simulation complete via $finish(1) at time 35722 NS + 0
./pat_stuck.v:2595 $finish;
ncsim> exit
```

Figure 18 SIMULATION COMPLETED WITH ZERO ERRORS

STEP 5: After doing simulation successfully, we have generated *.shm* file. This file is then opened on Simvison. We can see the waveforms for the respective design or block.

The figure 19 shows the screenshot of the waveforms shown on the tool. The left hand side shows the names of all the modules present. When we click on any module we can see the respective signals inside the module. And clicking on the signal we can see the respective waveforms for each signal. These waveforms are shown w.r.t time. Suppose we get an error at time = 30 ps while performing simulations so when we open this waveform for that particular signal on simvision and I can directly see the value at that particular time. We can adjust the time from left hand side top.

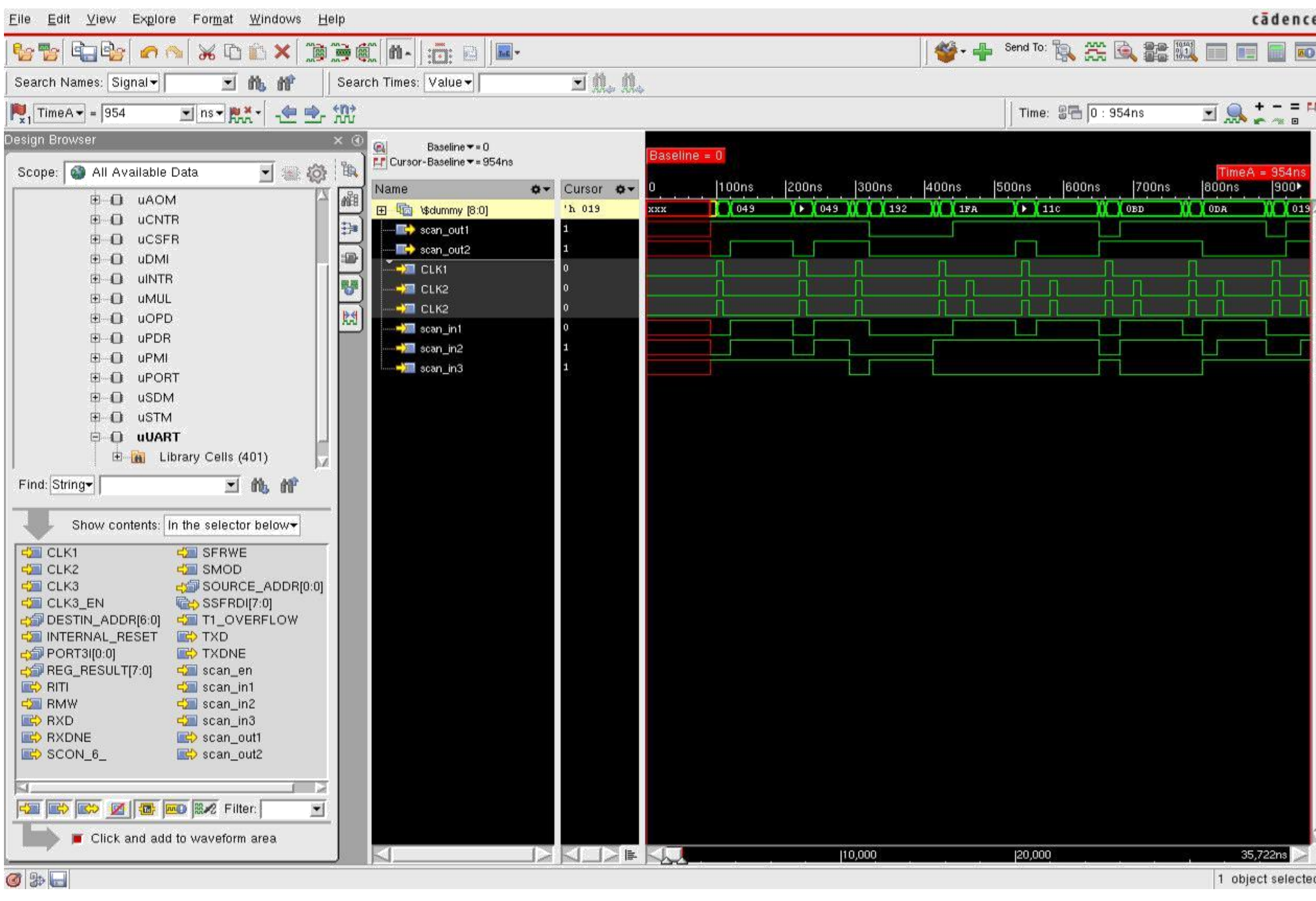


Figure 19 THE WAVEFORM SHOWING THE SIMULATED PATTERNS

<u>TOOL USED</u>: Ncsim version 15.20.45

CHAPTER -5

ISSUES OBSERVED AND RESULTS

As the process of Scan insertion, ATPG and Simulations was proceeding. I faced certain issues while completing these tasks. I want to discuss all those and also will also give the solutions for them. Getting on the positive side, whenever any issue was observed, I was studying or observing that with keen interest which also helped to develop my knowledge. So, I would say if problems are there, consider it as the chance to learn something new.

Scan insertion - In which we convert different flip flops into scan flip flops and then bind those scan flip flops into chains, this process is called scan insertion. The major error I got while doing scan insertion was an error named "S1 Violation".

S1 violation- Generally when we are connecting all the flops and making different scan chains, our task is mainly to connect all the flops together such that whenever we need to turn them ON or OFF we can do that together.S1 violation says when scan chains are added in the design and clocks are also added then whenever clock is set to low value then all the blocks in the design should stop working (as no clock input is present) and give no output at all. But in this case when the clock was set to low value, still one gate was shown in working condition.

This clearly set to S1 violation in the design.

Solution - When I checked the scan insertion script, I found that value of clock was not correctly defined in the script. And also, the version of Tessent tool used was not supporting it.

Then after using everything corrected in the design, the problem was resolved.

Also if we are not getting the exact solution for this, then we use a command called *"analyze_drc_violation <violation_number>"*. This will run the DRC analysis for the specific violation and allow the simulation data to be displayed. In the following example, the failure is

due to the clock pin "CP" being set to X. To pass the S1 DRC, the clock must be set to its off state of 0.

RESULTS- After all the processes are performed including Scan Insertion, ATPG and Simulations. As explained under the work done section the results are generated after every step. After the last step i.e. when Simulations are done and correct results are verified, we will send the design into the next step.

The main reason of DFT is to cover all the faults which could possibly occur in the design, so after all the steps are completed we check the fault coverage in the design. This fault coverage is also shared with the customer. Maximum is the fault coverage of the design more satisfied would be the customer. After the other steps are also followed according to the VLSI design flow, this fault coverage is calculated and shared by the company to the customer.

In DFT we also work to maximize the Fault coverage of the design, all the steps performed in the above sections also help in increasing fault coverage of the design.

CHAPTER – 6

CONCLUSION AND FUTURE SCOPE

DFT technology is now used from a long time period. As it is rightly said that an untested chip is a failed chip, so we have proved with our discussion that if a chip is untested and produced in the market, it will fail. This shows the importance of design for testability in the VLSI industry. The scan insertion process, ATPG processes, BIST are used widely. ATPG also has always been a research topic in the CAD design and testing environment.

While doing ATPG, the main focus is always on getting the maximum fault coverage for our design. The more is the fault coverage, more is the chip reliability. So in future, the different processes are suggested that could cover more number of faults and make chip more reliable.

Also during testing of the chip, sometimes some extra amount of current and power is generated which is a waste, so we should adopt certain techniques and method which could minimize this.

CHAPTER -7

REFERENCES

[1] Steven D. Millman and Edward J. McClusky for Bridging, Transition and stuck-open faults in self testing CMOS checkers, IEEE 1991.

[2] Keshava Iyengar Satish,Tutorial on Design for Testability (DFT):An ASIC Design Philosophy for testability from Chips to Systems, IEEE 1993.

[3] S.Barbagallo, M.Lobetti Bodoni and Medina, Scan insertion criteria for low design impact, IEEE 1996.

[4] Nikhil Sharma and C.P. Ravikumar, Improving Bridge fault testability and confidence of IDDQ testing through circuit placement, IEEE 1996.

[5] Hiroshi Yamazaki and Yukiya Miura, IDDQ testability of flip flop structures, IEEE 1996.

[6] L1. Ribas-Xirgo and J. Carrabina Bordoll, Automatic test pattern generation for IDDQ fault based upon symbolic simulation, IEEE 1996.

[7] Tsin-Yuan Chang and Weihong Chen, An IDDQ fault location Scheme, IEEE 1997.

[8] Yoshinobu higami, Toshiyuki Maeda and Kozo Kinoshita, Sequential circuit test generation for IDDQ testing of Bridging faults, IEEE 1997.

[9] Janak H. Patel, Stuck at fault- A fault model for next millennium, IEEE international test conference 1998.

[10] Wuudiann Ke and Khoan Truong, Design with testability for a platform based SOC design methodology, IEEE 1999.

[11] Eric Liau and Doris Schmitt- Landsiedel, evolution of automatic semiconductor test equipment: automatic test pattern learning, classification, optimization and generation for power supply noise, IEEE 2003.

[12] C.Metra, TM Mak and M.Omana, Are our design for testability features fault secure, Design, automatic and test in Europe conference and exhibition IEEE 2004.

[13] Seongmoon Wang and Srimat T. Chakradhar, A Scalable Scan-Path Test Point Insertion Technique to Enhance Delay Fault Coverage for Standard Scan Designs, IEEE 2006.

[14] Xiao-Xin FAN, Yu HU and Laung-Terng Wang, An On-Chip Test Clock Control Scheme for Multi-Clock At-Speed Testing, IEEE 2007.

[15] Irith Pomeranz and Sudhakar M.Reddy, Unspecified Transition Faults: A Transition Fault Model for At-Speed Fault Simulation and Test Generation, IEEE 2008.

thesis

ORIGINALITY REPORT

15%	11%	8%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

#	Source	%
1	manualzz.com Internet Source	3%
2	academic.odysci.com Internet Source	1%
3	doaj.org Internet Source	1%
4	www.ida.liu.se Internet Source	1%
5	www.computer.org Internet Source	1%
6	ir.lib.nthu.edu.tw Internet Source	1%
7	documents.mx Internet Source	1%
8	Submitted to Universiti Sains Malaysia Student Paper	1%
9	www.coursehero.com Internet Source	1%

10	Y. Higamit, T. Maeda, K. Kinoshita. "Sequential circuit test generation for IDDQ testing of bridging faults", Digest of Papers IEEE International Workshop on IDDQ Testing, 1997 Publication	<1%
11	xplorestaging.ieee.org Internet Source	<1%
12	fr.scribd.com Internet Source	<1%
13	K.I. Satish. "Tutorial on design for testability (DFT) "An ASIC design philosophy for testability from chips to systems"", Sixth Annual IEEE International ASIC Conference and Exhibit, 1993 Publication	<1%
14	www.eetimes.com Internet Source	<1%
15	E.J. McCluskey. "Bridging, transition, and stuck-open faults in self-testing CMOS checkers", [1991] Digest of Papers Fault-Tolerant Computing The Twenty-First International Symposium, 1991 Publication	<1%
16	scholar.uwindsor.ca Internet Source	<1%

Irith Pomeranz. "Unspecified Transition Faults:

Scanned by CamScanner

17	A Transition Fault Model for At-Speed Fault Simulation and Test Generation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 01/2008 Publication	<1%
18	Submitted to Visvesvaraya Technological University Student Paper	<1%
19	iq.opengenus.org Internet Source	<1%
20	bbs.hwrf.com.cn Internet Source	<1%
21	Submitted to Institute of Technology, Nirma University Student Paper	<1%
22	Submitted to Chandigarh University Student Paper	<1%
23	Nor Azura Zakaria, Edward V. Bautista Jr., Suhaimi Bahisham Jusoh, Weng Fook Lee, Xiaoqing Wen. "Case Studies on Transition Fault Test Generation for At-speed Scan Testing", 2010 IEEE 25th International Symposium on Defect and Fault Tolerance in VLSI Systems, 2010 Publication	<1%
24	"A Designer's Guide to Built-In Self-Test",	

Scanned by CamScanner

	Source	Similarity
	Springer Nature, 2002 Publication	<1%
25	Dong Xiang. "", IEEE Transactions on Circuits and Systems II Express Briefs, 5/2007 Publication	<1%
26	Submitted to Amity University Student Paper	<1%
27	Submitted to CSU, San Jose State University Student Paper	<1%
28	K.K. Saluja. "Sequential test generation with reduced test clocks for partial scan designs", Proceedings of IEEE VLSI Test Symposium VTEST-94, 1994 Publication	<1%
29	P.K. Kapur, Hoang Pham, A. Gupta, P.C. Jha. "Software Reliability Assessment with OR Applications", Springer Nature, 2011 Publication	<1%
30	en.wikipedia.org Internet Source	<1%
31	Submitted to Temple University Student Paper	<1%
32	Xiaoxia Wu. "Scan-chain design and optimization for three-dimensional integrated circuits", ACM Journal on Emerging	<1%

Scanned by CamScanner

Technologies in Computing Systems, 07/01/2009

Publication

33	Tasnim Ikra Rahman, Nishat Tasnim Liza, Bushra Bente Khair, Hamidur Rahman. "Fault diagnosis of differential and single stage amplifier utilizing combined effect of gain and phase of output voltage", 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), 2016 Publication	<1%
34	Submitted to University of Glasgow Student Paper	<1%

Exclude quotes On

Exclude bibliography On

Exclude matches < 3 words

Aman.

Scanned by CamScanner