MUTATION TESTING USING TIME-SHIFT OPERATOR

by

Hari Krishna Parimi, B. Tech

THESIS

Presented to the Faculty of

The University of Houston Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Computer Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

MAY, 2018

MUTATION TESTING USING TIME-SHIFT OPERATOR

by

Hari Krishna Parimi

APPROVED BY

_____

Ishaq Unwala, Ph.D., Chair

_____

Thomas Harman, Ph.D., Committee Member

_____

Hakduran Koc, Ph.D., Committee Member

APPROVED/RECEIVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

_____

Said Bettayeb, Ph.D., Associate Dean

_____

Ju H. Kim, Ph.D., Dean

Dedication

To

The Field of Computer Engineering and My family

Acknowledgements

I would like to express my sincere thanks, appreciation, and gratitude to all of those who have directly or indirectly contributed to my research work and those who have supported me throughout the entire process. I will always be grateful for that.

I would like to thank my supervisor Dr. Ishaq Unwala for his excellent guidance and engagement through my thesis period. He has been supportive since the day I began working on my research. His insightful discussions and suggestions on the research helped me finish this thesis successfully. Moreover, I would like to thank my thesis committee, Dr. Thomas Harman, and Dr. Hakduran Koc, for their encouragement and their helpful advice.

My immense love goes to my grandparents, Tataya (Grand Father) Ranga Rao Koduri, Amma (Mummy) Rama Lakshmi Parimi, Nana (Daddy) Venkata Ratnam Parimi, (Mavaya) Ganesh Koduri. I cannot thank them enough for their unconditional love, support and care through all these years. I would not have made it this far without them. Loads of love and thanks to my brother Naveen Krishna Parimi, who blossomed and cherished with me every great moment and supported me by keeping me harmonious and helping me putting pieces together.

Finally, I would like to thank my friends, roommates and my bayyas (brothers), Jayaram Immaneni, Rajesh Vangapalli, Saran, Shyam Karuparthy, Sudhakar Shivaram, Santosh Pruthvi for encouraging me all the time to get my thesis progressed and for a wonderful love they pour on me, making my life joyful.

ABSTRACT

MUTATION TESTING USING TIME-SHIFT OPERATOR

Hari Krishna Parimi
University of Houston-Clear Lake, 2018

Thesis Chair: Dr. Ishaq Unwala

Functional verification plays a critical role in ensuring that a digital integrated circuit (IC) meets the design specification. Dynamic verification uses a large number of test vectors. To analyze and improve the quality of these test vectors, the technique of mutation testing is used. In mutation testing the design is mutated with a known fault. To verify the quality of the test vectors the mutated design is retested with test vectors. Current mutation operators include arithmetic, logical, or relational operators. These operators mutate functional portion of the design. However, a significant number of design faults are related to signal timing. To mutate the design for signal timing, this research introduces a new operator, time-shift operator. Time-shift operator allows mutation of the signal timing, which allows an improvement in the quality of test vectors. In this research, it is shown that time-shift operator can be used in combination and sequential designs. This research also shows that the time-shift operator can be utilized in both behavioral and gate-level designs. The results of 9 different designs are presented covering all the cases of combination, sequential, behavioral and get-level designs.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1: INTRODUCTION

Functional verification is a critically important task in design of a digital integrated circuit (IC). The market demand for ICs with higher performance, lower power and smaller size is leading to increasing complexity as designer maximize the performance from every clock cycle while managing power consumption on larger transistor budgets.

In order to find design faults, i.e. deviation from specification, manufacturers employ a team of verification engineers. The most common verification technique used in industry is dynamic verification. Dynamic verification is based on using test vectors to simulate the IC design and observe the design for incorrect operation. Verification team generally generates a large set of test vectors based on the specification proved by the design architect. The design is simulated using an individual test vector. The design output, after an appropriate delay, is observed. If the design output matches the expected result the design is considered to have passed that particular test vector. If the design output does not match expected result, the design may have a fault or the test vector may be inappropriate. The cause of fault is resolved and test vector is reapplied, till the design passes the test vector. Similarly, all the test vectors are individually applied and outputs compared to the expected results. If the design passes all test vectors pass, the design is considered Golden design model, i.e. meeting the specification.

Dynamic simulation has very high dependency on the quality of test vectors. If the quality of the test vectors is poor then hidden faults in the design may not be exposed. In order to verify and improve the quality of the test vectors a technique known as mutation testing [4] is used.

Mutation testing involves modifying the Golden design model by introducing a known fault. The modified design is then called Mutated design model. The test vectors,

which the Golden design had previously passed, are then reapplied to the Mutated design to verify the quality of the test vectors.

Rest of this chapter discusses, the concept of mutation, the types of mutations based output results as well as the strength of mutation. Some general types of design concepts are also discussed.

Chapter 2 reviews the previous work related to mutation testing done by researchers.

Chapter 3 discusses the methodology of applying the mutation.

Chapter 4 provides the results from mutation of designs.

Finally, in Chapter 5 draws some conclusions from the research and discusses future work.

Reference section and appendix completed this document.

There is also a separate supplemental file containing the complete Verilog code used in the research.

1.1 Mutation

The process of mutation takes a Golden design and, after copying it as a separate file, inserts a known fault into the copy of the Golden design. This mutated copy is called Mutated design. Each Golden design can be mutated a number of times with each mutation being at different location in the design or a different type. These mutations can be tested individually or as a subset. It is common practice is to test each mutation individually to easily identify the strength or weakness in test vectors.

Mutations can be classified into equivalent mutate, and killed mutate [20]. If the Mutated model passes the same set of test vectors as Golden model without any failing

test vectors, then the mutation is said to be an equivalent mutant [20]. However if the Mutated design fails to pass on one or more of the test vectors, then the mutation is said to be a killed mutant [20].

1.1.1 Killed Mutant

If the output of the Golden design and the Mutated design are different for atleast one test in the set of test vectors then the mutantion is said to be a killed mutant [20].

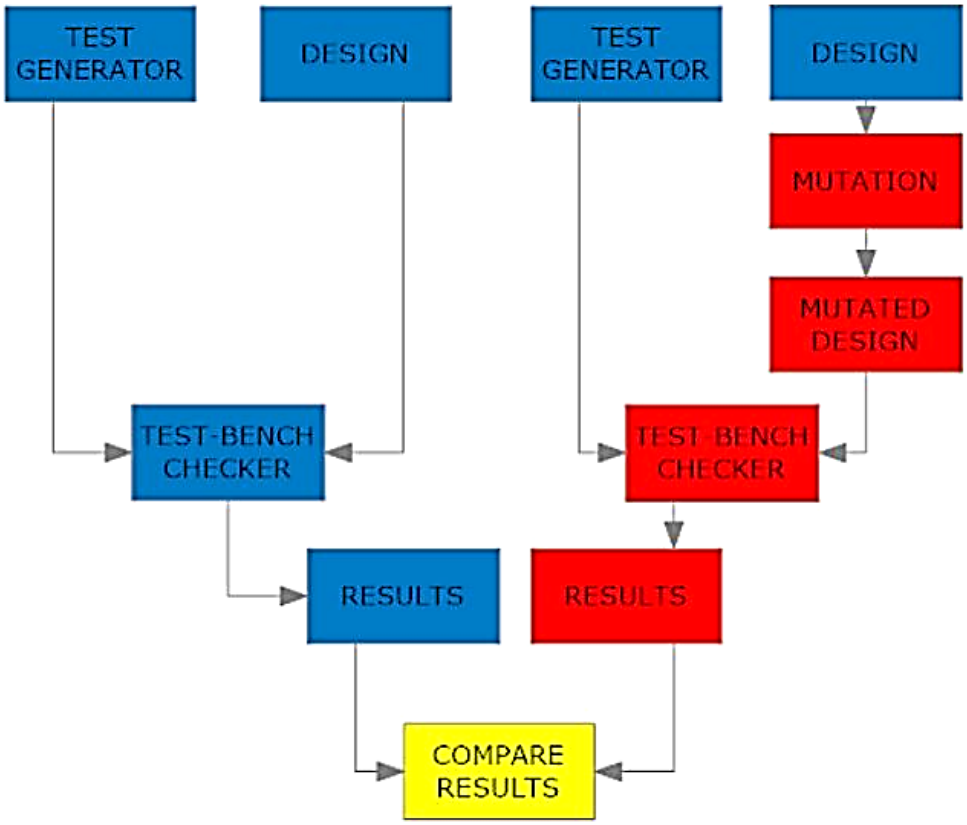
1.1.2 Equivalent Mutant

If the outputs of the Golden design and the Mutated design are same for all the tests in the set of test vectors then the mutantion is said to be an equivalent mutant [20].

1.2 Mutation Process

The mutation process is graphically presented in *Figure 1.1*

Figure 1.1
Mutation Process



The above process explains how the Golden design and the Mutated design are compared.

1.2.1 Test Generator

Tests generator module generates the set of test vectors to verify the design. The same tests are applied to both the Golden and the Mutated designs.

1.2.2 Test-Bench Checker

Test-bench checker module checks the output of the design with the expected result for the tests generated by the test generator. The test-bench checker can check both the Golden design and the Mutated design.

1.2.3 Results

Result Comparator compares the outputs from both the Golden design and Mutated design. If there is a mismatch then the applied test has successfully detected the mutation, otherwise the mutation was not detect.

1.3 Strength of Mutation

This section explains how strong the mutation is and defines the strength of the mutation.

1.3.1 Weak Mutation

Weak mutation is also known as Fault activation. In weak mutation, the test activates the fault at the fault location but does not propagate the faulty value, and thus cannot be detected at the output, then the mutation is said to be a weak mutation [20].

1.3.2 Firm Mutation

Firm mutation is also known as Fault propagation. In firm mutation, the test activates the fault at the fault location and propagates the fault towards the output, however fault progogation stops before reaching the output, and thus mutation cannot be detected, then the mutation is said to be a firm mutation [20].

1.3.3 Strong Mutation

Strong mutation is also known as Fault detection. In strong mutation, the tests activates the fault at the fault location and the fault is propagated all the way to the output, thus the mutation is detected, then the mutation is said to be a strong mutation [20].

1.4 Design

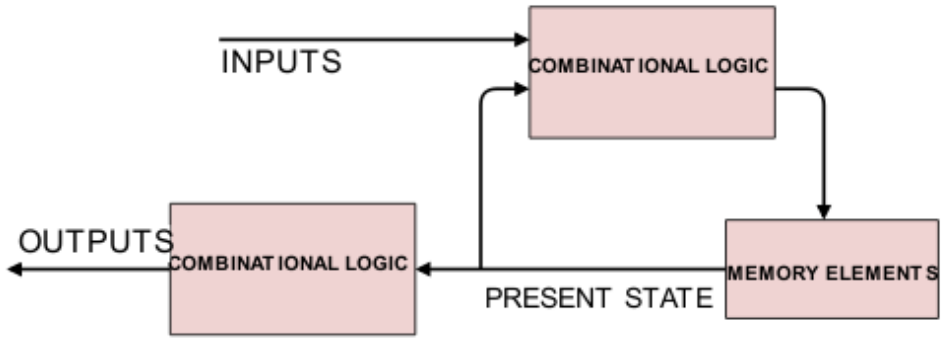
In order to test and demonstrate the effectiveness of the proposed problem statement (mutating by a time-shift operator) a number of designs were tested. For

sequential design, finite state machine (FSM) were tested. The states of the FSM are generally registers that are flip-flops. The FSM designs are classified as Moore or Mealy depending on their output relation with inputs and their current states.

1.4.1 Moore Design

If the outputs depend only on the current state, then the design is said to be Moore Design [22]. *Figure 1.2* shows the block-diagram for this type of design.

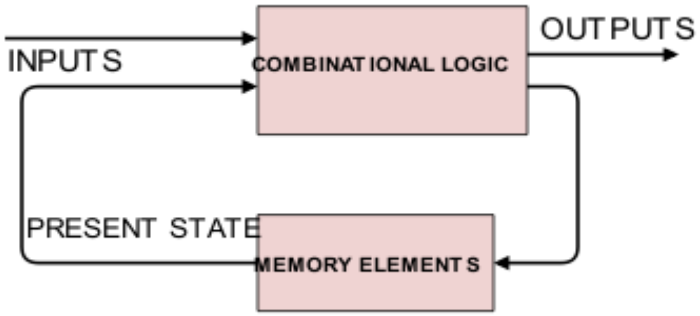
Figure 1.2
Moore Design[7]



1.4.2 Mealy Design

If the outputs depend on the current state and input, then the design is said to be Mealy Design [22]. *Figure 1.3* shows the block-diagram for this type of design.

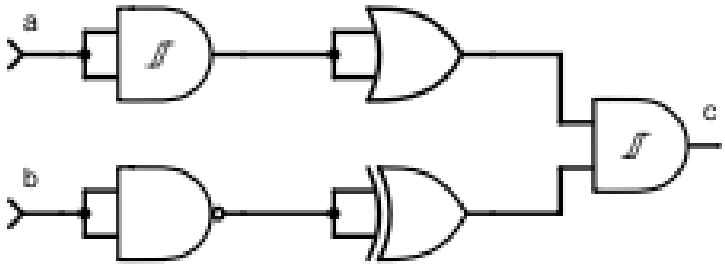
Figure 1.3
Mealy Design[8]



1.4.3 Golden Design

A design is called Golden design or Device Under Test (DUT) where the design is assumed to be faultless. *Figure 1.4* is an example we have taken as a Golden design with two AND gates, one OR gate, one XOR gate, one NAND gate and later mutated it further as shown in *Figure 1.5* to make it a Mutated design.

Figure 1.4
Golden Design



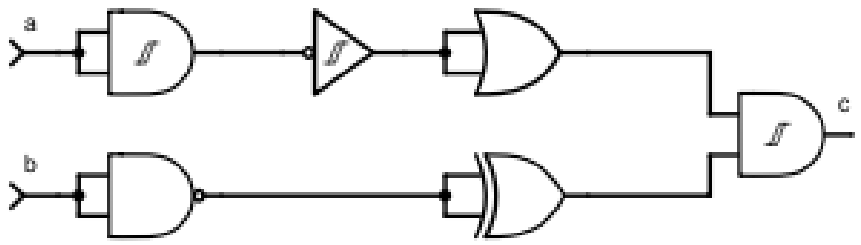
1.4.4 Mutated Design

A design is said to be mutated if there is any fault added to the original design. The fault consist of adding, removing or even changing a portion of the Golden design

with a known fault. Presently, mutation of designs is done by changing the airthmetic operators [17] in the design.

Figure 1.4 is mutated with a not gate that can be seen in *Figure 1.5* to make it a mutated one.

Figure 1.5
Mutated Design



1.5 D-FF

D-FF is a D-flipflop. *Figure 1.6* shows the inputs and outputs for the D-FF. *Table 1* is the truth table for the D-FF. In our approach, we used D-FF as the time shift operator or as our main mutation concept.

Figure 1.6
D-FF

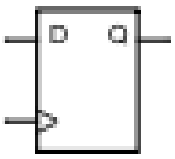


Table 1
D-FF Truth Table

Clk	D	Q	~Q
0	X	No change	No change
1	0	0	1
1	1	1	0

Clk is Clock, d is the data which is input. Q and ~Q are the outputs. x is a don't care value. From the *Table 1*, we can understand that if clock is low (that is 0) the output has no change in its value. If the clock is high (that is 1) the output is same as its input.

1.6 Time-shift Operator

The operator which shifts the time is termed as time-shift operator.. By adding or removing the time-shift operator we can speed-up or slow-down the signal. The next sub-sections 1.6.1 and 1.6.2 will show how the timing is changed by use of time-shift operator with an example. The device we use as a time-shift operator is D-FF, which was discussed in section 1.5.

1.6.1 Operator Added

If we add a time-shift operator to the design the signal time gets delayed. In the design below, a D-FF is used which delays the signal timing by one clock cycle. *Figure 1.7* is taken as an example and a D-FF is added before the output O1 as shown in *Figure 1.9*. The waveforms of the Design_1 which is *Figure 1.8* when compared to the waveform when the operator is added to the Design_1 can be seen in *Figure 1.10* which indicates the delay of the signal by a clock cycle.

Figure 1.7
Example Design_1

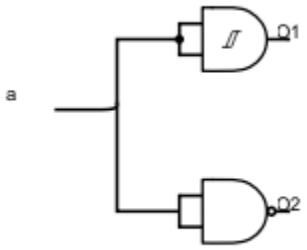


Figure 1.8
Waveform Design_1

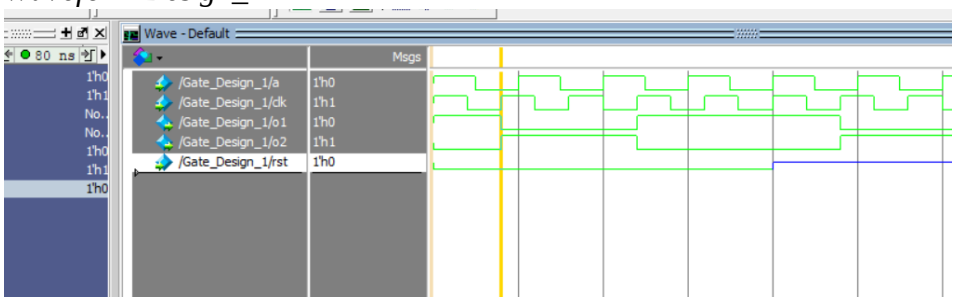


Figure 1.9
Example Design_1_D-FF

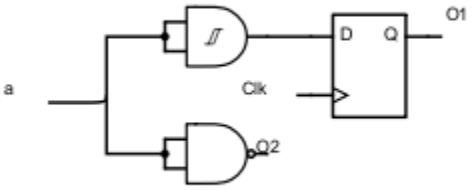
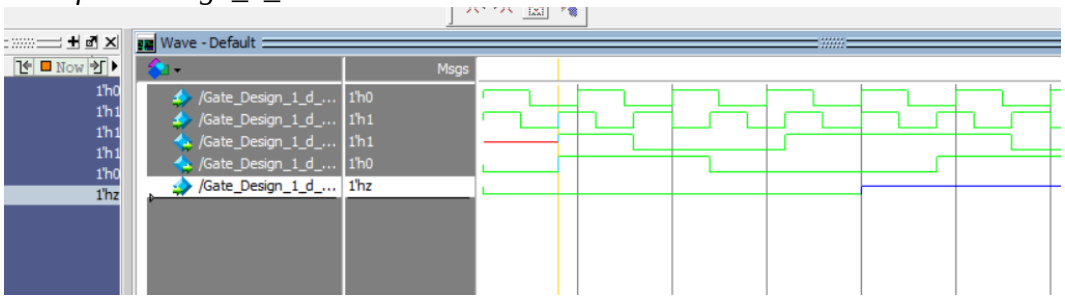


Figure 1.10
Waveform Design_1_D-FF



1.6.2 Operator Detached

To speed-up signal timing, a D-FF needs to be removed. Note that the D-FF can only be removed if it is already present in the Golden Design. In the following example a D-FF is removed, which speeds up the signal timing by one clock cycle. *Figure 1.11* is taken as an example and a D-FF is detached before the output O1 as shown in *Figure 1.13*. The waveforms of the Design_2 which is *Figure 1.12* when compared to the waveform when the operator is detached to the Design_2 can be seen in *Figure 1.14* which indicates the advancement of the signal by a clock cycle.

Figure 1.11
Example Design_2

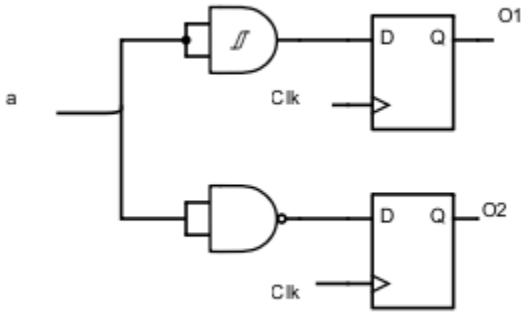


Figure 1.12
Waveform Design_2

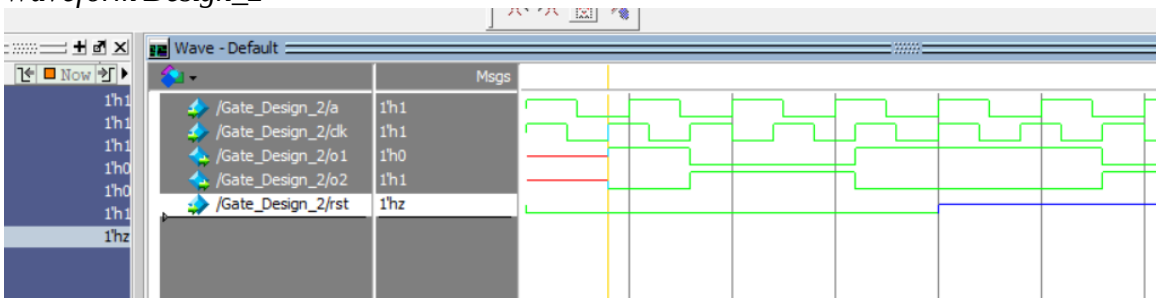


Figure 1.13
Example Design_2_D-FF_Detached

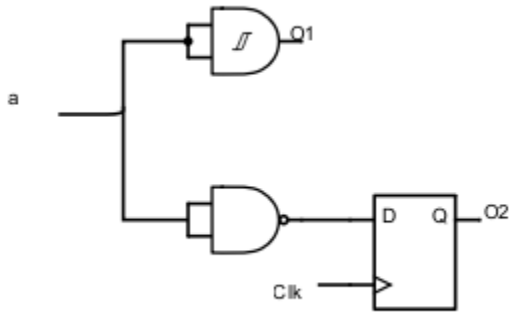
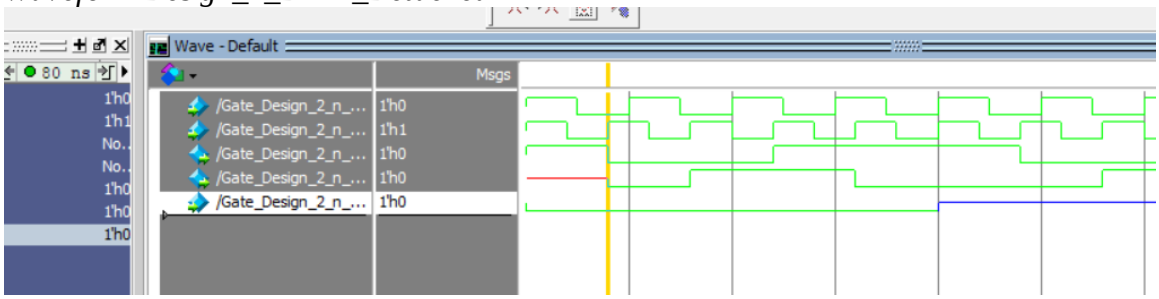


Figure 1.14
Waveform Design_2_D-FF_Detached



CHAPTER 2: REVIEW OF LITERATURE

Lisherness, et al. [1] in their paper mainly discuss the method used for VHDL RTL functional verification. In the paper, the authors propose an approach to reduce the simulation time based on FPGA emulation. A Meta-Mutant Test bench is used to emulate mutants. The type of mutation testing is explained with some illustrations. The authors tried to prove that this type of hardware and software-based emulation is 20 times quicker than the regular software-based simulations. The paper and their results prove that their approach is more than 10 times faster. In the future ideas section, the authors are focusing on stimuli generation in extension to their work to improve the functional verification quality while the present work focuses on the optimization of time. This paper made us think of optimizing the time by mutation.

Xie, et al.'s [2] paper is based on Markov-chain and mutants. This paper provides good ideas for generating tests and describes how to calculate the mutants and help us explore more about mutants.

Bombieri, et al.'s [3] paper explains the importance of software testing for mutation. The paper explains how to inject the mutants and how the verification can be done in a systematic way.

Huang, et al.'s [4] paper explains the principles of generating a test-bench. This paper discusses research on test-bench qualification with the help of mutation. This paper gave us the idea for a quality test-bench.

From Serrestou, et al.'s [5] paper, we got to know how the fault in the mutation can be activated, propagated and then detected in the whole process.

Lin, et al.'s [6] paper tells us about firm mutation, living mutants (a mutant that is activated initially and propagated to the final output) and how they are being decreased with a probabilistic approach using Error Propagation Analysis (EPA) tools.

Nguyen, et al.'s [17] paper tells us how the mutants and the test data can be generated in an effective way in the software and implemented into the hardware to make the test cost effective.

Bombieri, et al.'s [18] paper explains mutation analysis for the Transaction Level Modeling(TLM) design. The mutation analysis is done for software testing, to find the quality test-bench. This paper explains how the mutations can be done and the definition of mutant, mutation and how to do mutation analysis.

Hsiao, et al.'s [19] paper explains the automated test generation and mainly we learned how to apply our mutation so that it can have a better chance of activation and propagation.

Jia and Harman's [20] paper teaches us how the mutation evolves, and the types of research that has been done over the decades. It also explains how the mutation can be done in various ways and the problems after a mutation is done and how to do mutation analysis in an effective manner. This paper made us to think how our mutation can be done and how we can do mutation analysis differently.

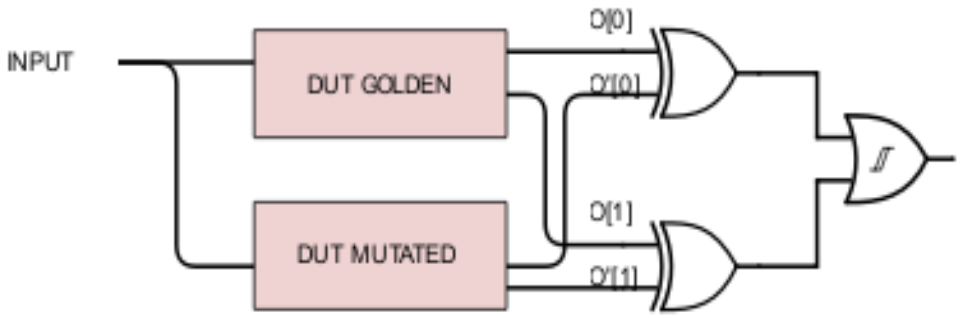
CHAPTER 3: METHODOLOGY

To conduct experiment of designs, Verilog language was chosen. These Verilog designs needed to be simulated with test vectors. To simulate the Verilog design, the student edition ModelSim simulator, by Mentor Graphics, was used. All the simulation results, including all waveform pictures are from the ModelSim simulator.

The general methodology is depicted in the *Figure 3.1*, the same inputs are provided to both the Golden and Mutated designs shown in the *Figure 3.1* as two different DUTs (Device Under Test). To check if both the designs have the same or different outputs, the corresponding output signals are passed through the ex-or gates. All the outputs of the EXOR gates are passed through an OR gate. If the corresponding outputs differ, the EXOR result is a 1. The result of the OR gate will be 1 if any one of the EXOR output is 1. This allows detection of mismatch due to the mutation.

Here, the design is mutated by adding a D-FF or detaching the D-FF. The D-FF is inserted or removed from the Golden design in random manner, no particular methodology or algorithm was used.

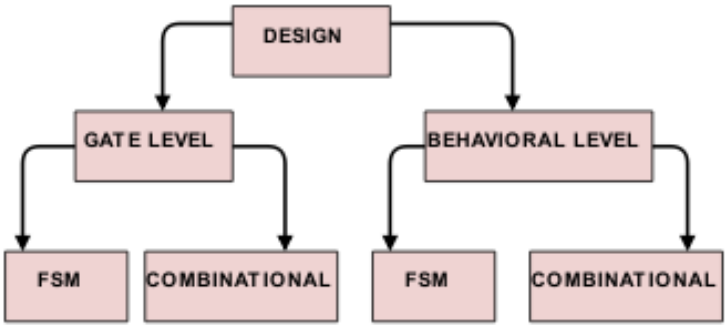
Figure 3.1
Methodology for Mutation Detection



3.1 Design

The designs used in this research can be categorized into four classes as shown in *Figure 3.2*. The Gate-level and Behavioral level designs which are divided into Finite state machines and combinational parts. Here, all the designs are made in System Verilog using model-sim student edition software. Even the test-benches made to verify the designs are made in System Verilog, using the same model-sim student edition software.

Figure 3.2
Design Classification



The Mutation can be done by adding or removing a flip-flop in all the designs, but the flip-flop can be removed only from the FSMs as there are no flip-flops in the combinational designs.

First, we will discuss a prototype vending machine design which is a Mealy design.

Elevator and simple-state transitions designs are like Moore design, which are explained in the later sections.

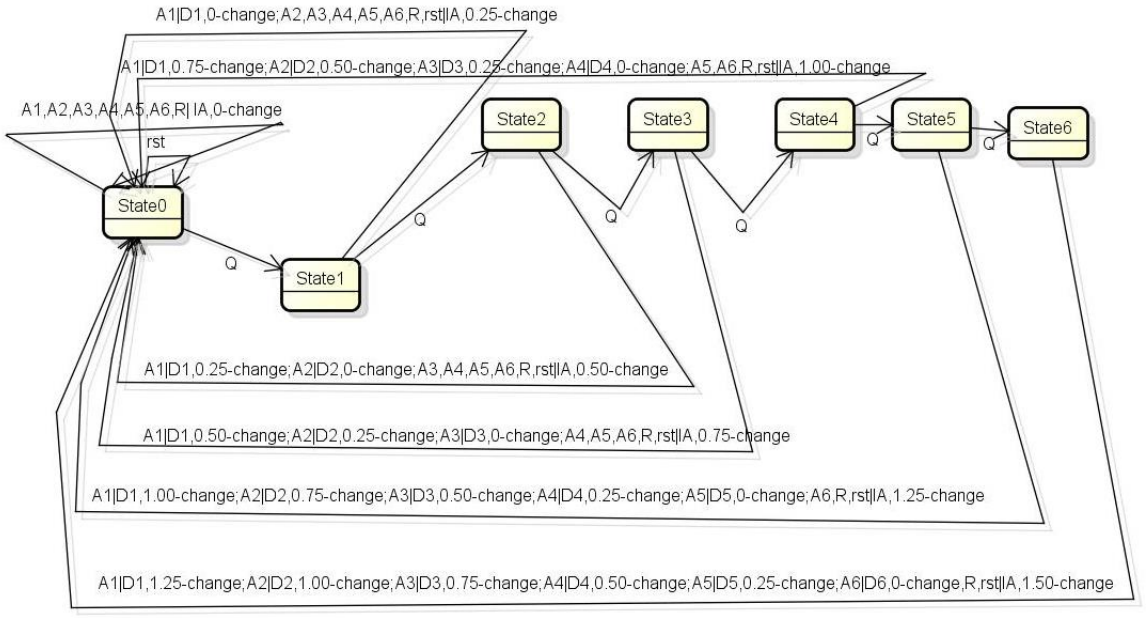
3.1.1 Vending Machine

The first design is a Vending Machine, which accepts only quarters and returns only quarters. It has total 8 inputs, one input is entering quarters one at a time into the machine, and 7 input buttons where one of the buttons is for the return of change at any time of the process, and 6 for the selection of 6 different items A1, A2, A3, A4, A5 and A6 with a cost of 0.25, 0.50, 0.75, 1.00, 1.25 and 1.50 respectively. It will give us an output as a change in quarters, "insufficient amount" is displayed, and dispenses items A1, A2, A3, A4, A5, and A6.

The *Figure 3.3* in the form of a state diagram shows how the whole process takes place. The notations on the transition arrows from one state to another are corresponding to Inputs|Outputs.

Figure 3.3 shows the state transition for the vending machine design used in this thesis.

Figure 3.3
Vending Machine State Transition



This state transition gives a clear picture of how the state transitions take place with the inputs and the outputs obtained. The Q in the *Figure 3.3* represents the inserted quarter. On the transition line (a connecion between the states), the input selected, and the corresponding output obtained is separated by "|". This design is used for evaluation of the methodology in both behavioral and Gate Level designs.

There are seven states with the following:

Inputs: $0.25 (The machine can accept quarters up to an amount of $1.50; this contains about 6 different states transiting from one state to another for every $0.25), and buttons or selections: - A1 (item which costs $0.25), A2 (item which costs $.50), A3 (item which costs $.75), A4 (item which costs $1.00), A5 (item which costs $1.25), A6 (item which costs $1.50), Return Change (R)

Outputs: Change (C - amount will be dispensed only in quarters which is a 3-bit), A1, A2, A3, A4, A5, A6 (six different items) represented as D (item dispensed which is a 3-bit).

Also, reset (rst) and clock (clk) are used.

At any stage of the process if rst button is pressed the change or amount that was deposited up to that point of time will be dispensed and system will go to the state S0.

The transition from S0:

Initially, we are at S0 (state zero – initial state).

If A1, A2, A3, A4, A5, A6 are selected we will get the output "insufficient amount" and system stays in state S0.

If rst or R is chosen, system will stay in this state S0.

If $0.25 is added by the user, system transits to state S1($0.25).

The transition from S1:

Now, we are at S1.

If A1 is selected A1 is dispensed and system will go to S0.

If A2, A3, A4, A5, or A6 are selected we will get a change of $0.25 and the output "insufficient amount" is displayed, and system will go back to state S0.

If rst or R is chosen, system will give a change of $0.25 and will go to state S0.

If $0.25 is added by the user, system transits to state S2 ($0.50).

The transition from S2:

Now, we are at S2.

If A1 is selected A1 is dispensed with change of $0.25 and it will go to S0.

If A2 is selected A2 is dispensed and it will go to S0.

If A3, A4, A5, or A6 are selected we will get change of $0.50 and the output "insufficient amount" is displayed, and it will go back to state S0.

If rst or R is chosen, it will give change of $0.50 and will go to state S0.

If $0.25 is added by the user, it transits to state S3 ($0.75).

The transition from S3:

Now, we are at S3.

If A1 is selected A1 is dispensed with change of $0.50 and it will go to S0.

If A2 is selected A2 is dispensed with change of $0.25 and it will go to S0.

If A3 is selected A3 is dispensed and it will go to S0.

If A4, A5, or A6 are selected we will get change of $0.75 and the output "insufficient amount" is displayed, and it will go back to state S0.

If rst or R is chosen, it will give a change of $0.75 and will go to state S0.

If $0.25 is added by the user, it transits to state S4 ($1.00).

The transition from S4:

Now, we are at S4.

If A1 is selected A1 is dispensed with change of $0.75 and it will go to S0.

If A2 is selected A2 is dispensed with change of $0.50 and it will go to S0.

If A3 is selected A3 is dispensed with change of $0.25 and it will go to S0.

If A4 is selected A4 is dispensed and it will go to S0.

If A5, or A6 are selected we will get change of $1.00 and the output "insufficient amount" is displayed, and it will go back to state S0.

If rst or R is chosen, it will give a change of $1.00 and will go to state S0.

If $0.25 is added by the user, it transits to state S5 ($1.25).

The transition from S5:

Now, we are at S5.

If A1 is selected A1 is dispensed with change of $1.00 and it will go to S0.

If A2 is selected A2 is dispensed with change of $0.75 and it will go to S0.

If A3 is selected A3 is dispensed with change of $0.50 and it will go to S0.

If A4 is selected A4 is dispensed with change of $0.25 and it will go to S0.

If A5 is selected A5 is dispensed and it will go to S0.

If A6 is selected it will get change of $1.25 and the output "insufficient amount" is displayed, and it will go back to state S0.

If rst or R is chosen, it will give change of $1.25 and go to state S0.

If $0.25 is added by the user, it transits to state S6 ($1.50).

The transition from S6:

Now, we are at S6.

If A1 is selected A1 is dispensed with change of $1.25 and it go to S0.

If A2 is selected A2 is dispensed with change of $1.00 and it go to S0.

If A3 is selected A3 is dispensed with change of $0.75 and it go to S0.

If A4 is selected A4 is dispensed with change of $0.50 and it go to S0.

If A5 is selected A5 is dispensed with change of $0.25 and it go to S0.

If A6 is selected A6 is dispensed and goes back to its ideal state S0.

If rst or R is chosen, it will give change of $1.50 and go to state S0.

If $0.25 is been added by the user, it will transit to S0 and change of $1.75 is returned.

Table 3.1
State Assignment

State Assignment($Q_2Q_1Q_0$)		
S0	0	000
S1	0.25	001
S2	0.50	010
S3	0.75	011
S4	1.00	100
S5	1.25	101
S6	1.50	110

Above *Table* 3.1 depicts how the states are assigned S0 to S6 as 3-bits ($Q_2Q_1Q_0$).

Table 3.2
Item Assignment

Item Assignment ($D_2D_1D_0$)	
A1/D1	001
A2/D2	010
A3/D3	011
A4/D4	100
A5/D5	101
A6/D6	110

Above *Table 3.2* depicts how the items as inputs (A_1 to A_6) and output items dispensed (D_1 to D_6) are assigned as ($D_2D_1D_0$).

Table 3.3
Next State Table

State Diagram								
P.S.	Next State ($Q^+_2\ Q^+_1\ Q^+_0$)							
(Q2Q1Q0)	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0 0	0 0 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	0 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 1 0	0 1 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 1 1	1 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
1 0 0	1 0 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
1 0 1	1 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
1 1 0	1 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0

The *Table 3.3* explains how the next states are obtained with the current states and the inputs.

Next State Equations:

With the help of *Table 3.3* and [9], we obtained the next state equations in terms of inputs and current state.

$Q_0^+ = A'.\ C'.\ E'.\ F'.\ G' + B'.\ C'.\ E'.\ F'.\ G'$ ($A - Q_2$; $B - Q_1$; $C - Q_0$; $D - X_3$; $E - X_2$; $F - X_1$; $G - X_0$)

$Q_1^+ = B'.\ C.\ E'.\ F'.\ G' + B.\ C'.\ E'.\ F'.\ G'$

$Q_2^+ = A.\ E'.\ F'.\ G' + B.\ C.\ E'.\ F'.\ G'$

Table 3.4
Output Table

P.S. (Q2Q1Q0)	Output (D_2 D_1 D_0 C_2 C_1 C_0) for Inputs ($X_3X_2X_1X_0$)							
	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0 0	0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0
0 0 1	0 0 0 0 0 0	0 0 1 0 0 0	0 0 0 0 0 1	0 0 0 0 0 1	0 0 0 0 0 1	0 0 0 0 0 1	0 0 0 0 0 1	0 0 0 0 0 1
0 1 0	0 0 0 0 0 0	0 0 1 0 0 1	0 1 0 0 0 0	0 0 0 0 1 0	0 0 0 0 1 0	0 0 0 0 1 0	0 0 0 0 1 0	0 0 0 0 1 0
0 1 1	0 0 0 0 0 0	0 0 1 0 1 0	0 1 0 0 0 1	0 1 1 0 0 0	0 0 0 0 1 1	0 0 0 0 1 1	0 0 0 0 1 1	0 0 0 0 1 1
1 0 0	0 0 0 0 0 0	0 0 1 0 1 1	0 1 0 0 1 0	0 1 1 0 0 1	1 0 0 0 0 0	0 0 0 1 0 0	0 0 0 1 0 0	0 0 0 1 0 0
1 0 1	0 0 0 0 0 0	0 0 1 1 0 0	0 1 0 0 1 1	0 1 1 0 1 0	1 0 0 0 0 1	1 0 1 0 0 0	0 0 0 1 0 1	0 0 0 1 0 1
1 1 0	0 0 0 0 0 0	0 0 1 1 0 1	0 1 0 1 0 0	0 1 1 0 1 1	1 0 0 0 1 0	1 0 1 0 0 1	1 1 0 0 0 0	0 0 0 1 1 0

The *Table 3.4* explains how the outputs are obtained with the current states and the inputs.

Output Equations:

With the help of *Table 3.4* and [9], we obtained the next state equations in terms of inputs and current state.

D2= A.E.F'.G' + A.C.E.F' + A.B.E.F' + A.B.E.G'

D1= A.E'.F + B.E'.F.G' + B.C.E'.F + A.B.F.G'

D0= A.E’.G + C.E’.F’.G + B.E’.F’.G + B.C.E’.G + A.C.F’.G + A.B.F’.G

C2=

A.B’.E.F + A.E.F.G + A.B’.C’.E.G + A.C.E’.F’.G + A.B.E’.F’.G + A.B.E’.F.G’

C1=

A’.B.C’.E + A.C.E’.F + B.C’.E’.F.G + B.C’.E.F’.G’ + B.C.E’.F’.G + B.C.E’.F.G’ + A.B’.E’.F.G’ + A.B’.C’.E’.F’.G + B.E.F.G

C0=

C.F.G’ + A’.C.E + C.E.G’ + C.E.F + A’.B’.C.F + A.C’.E’.G + A.B.F’.G + B.C’.E’.F’.G

3.1.2 Simple State Transition

Figure 3.4
Simple State Transition

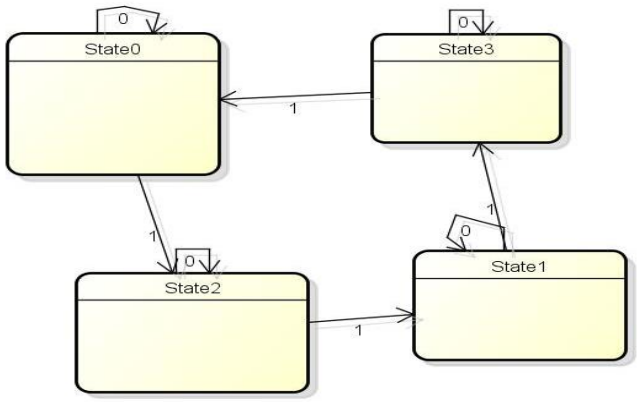


Figure 3.4 explains the state transition for simple state transition design with the input. This design is used for evaluation of the methodology in both behavioral and Gate Level designs.

There are a total four states with the following:

Inputs: x_in is the only one input.

Outputs: out which is the state in which the system is present.

At any stage of the process if rst button is pressed the out would be state 0.

The transition from S0:

Initially, we are at S0 (state zero – initial state).

If x_in is 1, we will get an output state 2.

If x_in is o, we will get an output state 0.

If rst is chosen, system will be in state S0.

The transition from S1:

Now, we are at S1.

If x_in is 1, we will get an output state 3.

If x_in is o, we will get an output state 1.

If rst is chosen, system will be in state S0.

The transition from S2:

Now, we are at S2.

If x_in is 1, we will get an output state 1.

If x_in is o, we will get an output state 2.

If rst is chosen, system will be in state S0.

The transition from S3:

Now, we are at S3.

If x_in is 1, we will get an output state 0.

If x_in is o, we will get an output state 3.

If rst is chosen, system will be in state S0.

Table 3.5
Simple state transition next state and output table

Input	Current State		Next State		Output
x_in	A	B	A	B	O
0	0	0	0	0	0 0
0	0	1	0	1	0 1
0	1	0	1	0	1 0
0	1	1	1	1	1 1
1	0	0	1	0	1 0
1	0	1	1	1	1 1
1	1	0	0	1	0 1
1	1	1	0	0	0 0

From the above *Figure 3.4 and Table 3.5*, we can infer that if the input x_in is given then the state transition takes place else system will be in its own state.

3.1.3 Elevator

Figure 3.5
Elevator State Transition

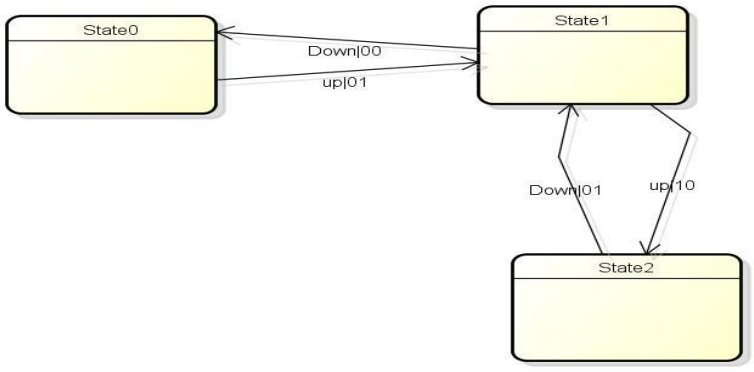


Figure 3.5 explains the state diagram for elevator design with the inputs up and down. This design is used for evaluation of the methodology in behavioral Level designs.

There are a total three states with the following:

Inputs: up and down are the inputs.

Outputs: out, which is the state in which system is present.

At any stage of the process if rst button is pressed the out would be state 0.

There are total of 3 states.

The transition from S0:

Initially, we are at S0 (state zero – initial state).

If up is 1, we will get an output state 1.

down can not be selected.

If rst is chosen, system will be in state S0.

The transition from S1:

Now, we are at S1.

If up is 1, we will get an output state 2.

If down is 1, we will get an output state 0.

If rst is chosen, system will be in state S0.

The transition from S2:

Now, we are at S2.

Up cannot be selected.

If down is 1, we will get an output state 1.

If rst is chosen, system will be in state S0.

Table 3.6
Elevator next state and output table

Input		Current State		Next State		Output
Up	Down	A	B	A	B	O
1	0	0	0	0	1	0 1
1	0	0	1	1	0	1 0
0	1	0	1	0	0	0 0
0	1	1	0	0	1	0 1

From the above *Figure 3.5 and Table 3.6*, we can infer that if the input up is given then the state transition takes place to the next state if down is selected system goes to the previous state.

CHAPTER 4: RESULTS

The circuit methodology discussed in Chapter 3 was followed to build a complete testing harness. Each of the nine Golden designs was mutated a number of times. Each mutation was tested individually with appropriate test vectors. Multiple mutations were not tested as a subset. The output results of each test, where the mutation was detected are show as 1 in the Table columns. If the output results of the test did not detect the mutation a 0 is shown in that column.

Detected in Middle:

If we have a weak or firm mutation the result is presented in column, "Detected in Middle". This is the OR value for the outputs obtained in the process of obtaining the end outputs. This is where the mutation has been detected in the middle of the design, but not at the outputs.

Detected at End:

If we have a strong mutation the result is presented in column, "Detected at End". This is the OR value obtained for the end outputs. This is where the mutation has been detected at the end of the process at the output.

4.1 Gate Level

Here, the designs are made in the gate level code using model-sim in system Verilog.

4.1.1 Gate Level FSM

4.1.1.1 Vending Machine

The *Table 4.1* shows the mutation results for four mutations tested with the same tests as shown in the left most column:

1)Mutation m0:

In this mutation, we added a D-FF to the (Present State) PS [0] for Vending Machine design.

2) Mutation m1:

In this mutation, we added a D-FF to the PS [1] for Vending Machine design.

3) Mutation m2:

In this mutation, we added a D-FF to the D [0] for Vending Machine design.

4) Mutation m3:

In this mutation, we removed a D-FF from the PS [0] for Vending Machine design.

Here, in Test Line S0_1 means input 1 is selected in the state S0. 2,3,4,5,6, R, Q are for input 2,3,4,5,6, Return and Quarter respectively. Total there are states S0, S1, S2, S3, S4, S5, S6. State S0 has only two lines in the text file used as a test vector. First line for the selection of inputs and the second line for the output. State S1 has three lines, as the first line for a quarter inserted, second line is for the input chosen and the third line is for the output obtained. Similarly, one extra line is added in the test vector as the state moves on. The extra line is for the addition of quarter. So, detected in middle values shown in *Table 4.1* increases from state to state.

Table 4.1
Vending Machine Gate Level FSM Mutation Results

Test Line	Mutation (m0)		Mutation (m1)		Mutation (m2)		Mutation (m3)	
	Detected in Middle	Detected at end	Detected in Middle	Detected at end	Detected in Middle	Detected at end	Detected in Middle	Detected at end
S0_1	0	0	0	0	0	0	0	0
S0_2	0	0	0	0	0	0	0	0
S0_3	0	0	0	0	0	0	0	0
S0_4	0	0	0	0	0	0	0	0
S0_5	0	0	0	0	0	0	0	0
S0_6	0	0	0	0	0	0	0	0
S0_R	0	0	0	0	0	0	0	0
S0_Q	0	0	0	0	0	0	0	0
S1_1	0,0	0	0,0	0	0,0	0	0,0	0
S1_2	0,0	0	0,0	0	0,0	0	0,0	0
S1_3	0,0	0	0,0	0	0,0	0	0,0	0
S1_4	0,0	0	0,0	0	0,0	0	0,0	0
S1_5	0,0	0	0,0	0	0,0	0	0,0	0
S1_6	0,0	0	0,0	0	0,0	0	0,0	0
S1_R	0,0	0	0,0	0	0,0	0	0,0	0
S1_Q	0,0	1	0,0	0	0,0	1	0,0	1
S2_1	0,0,1	0	0,0,0	0	0,0,1	0	0,0,1	0
S2_2	0,0,1	0	0,0,0	0	0,0,0	0	0,0,1	0

Test Line	Mutation (m0)		Mutation (m1)		Mutation (m2)		Mutation (m3)	
	Detected in Middle	Detected at end	Detected in Middle	Detected at end	Detected in Middle	Detected at end	Detected in Middle	Detected at end
S2_3	0,0,1	0	0,0,0	0	0,0,0	0	0,0,1	0
S2_4	0,0,1	0	0,0,0	0	0,0,0	0	0,0,1	0
S2_5	0,0,0	0	0,0,0	0	0,0,0	0	0,0,0	0
S2_6	0,0,1	0	0,0,0	0	0,0,0	0	0,0,1	0
S2_R	0,0,1	0	0,0,0	0	0,0,0	0	0,0,1	0
S2_Q	0,0,0	1	0,0,0	1	0,0,0	1	0,0,0	1
S3_1	0,0,0,1	0	0,0,0,1	0	0,0,0,1	1	0,0,0,1	0
S3_2	0,0,0,1	0	0,0,0,1	0	0,0,0,0	0	0,0,0,1	0
S3_3	0,0,0,1	0	0,0,0,1	0	0,0,0,0	0	0,0,0,1	0
S3_4	0,0,0,1	0	0,0,0,1	0	0,0,0,0	0	0,0,0,1	0
S3_5	0,0,0,1	0	0,0,0,1	0	0,0,0,0	0	0,0,0,1	0
S3_6	0,0,0,1	0	0,0,0,1	0	0,0,0,0	0	0,0,0,1	0
S3_R	0,0,0,1	0	0,0,0,1	0	0,0,0,0	0	0,0,0,1	0
S3_Q	0,0,0,0	1	0,0,0,0	1	0,0,0,0	1	0,0,0,0	1
S4_1	0,0,0,0,1	0	0,0,0,0,1	0	0,0,0,0,1	1	0,0,0,0,1	0
S4_2	0,0,0,0,1	0	0,0,0,0,1	0	0,0,0,0,0	0	0,0,0,0,1	0
S4_3	0,0,0,0,1	0	0,0,0,0,1	0	0,0,0,0,1	1	0,0,0,0,1	0
S4_4	0,0,0,0,1	0	0,0,0,0,0	0	0,0,0,0,0	0	0,0,0,0,1	0
S4_5	0,0,0,0,0	0	0,0,0,0,0	0	0,0,0,0,0	0	0,0,0,0,0	0

Test Line	Mutation (m0)		Mutation (m1)		Mutation (m2)		Mutation (m3)	
	Detected in Middle	Detected at end	Detected in Middle	Detected at end	Detected in Middle	Detected at end	Detected in Middle	Detected at end
S4_6	0,0,0,0,1	0	0,0,0,0,0	0	0,0,0,0,0	0	0,0,0,0,1	0
S4_R	0,0,0,0,1	0	0,0,0,0,1	0	0,0,0,0,0	0	0,0,0,0,1	0
S4_Q	0,0,0,0,0	1	0,0,0,0,0	1	0,0,0,0,0	1	0,0,0,0,0	1
S5_1	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,1	1	0,0,0,0,0,1	0
S5_2	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,0	0	0,0,0,0,0,1	0
S5_3	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,1	1	0,0,0,0,0,1	0
S5_4	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,0	0	0,0,0,0,0,1	0
S5_5	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,0	0	0,0,0,0,0,1	0
S5_6	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,0	0	0,0,0,0,0,1	0
S5_R	0,0,0,0,0,1	0	0,0,0,0,0,1	0	0,0,0,0,0,0	0	0,0,0,0,0,1	0
S5_Q	0,0,0,0,0,0	1	0,0,0,0,0,0	1	0,0,0,0,0,0	1	0,0,0,0,0,0	1
S6_1	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	1	0,0,0,0,0,0,1	0
S6_2	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,0	0	0,0,0,0,0,0,1	0
S6_3	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	1	0,0,0,0,0,0,1	0
S6_4	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,0	0	0,0,0,0,0,0,1	0
S6_5	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	1	0,0,0,0,0,0,1	0
S6_6	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,0	0	0,0,0,0,0,0,1	0
S6_R	0,0,0,0,0,0,1	0	0,0,0,0,0,0,1	0	0,0,0,0,0,0,0	0	0,0,0,0,0,0,1	0
S6_Q	0,0,0,0,0,0,0	1	0,0,0,0,0,0,0	1	0,0,0,0,0,0,0	1	0,0,0,0,0,0,0	1

4.1.1.2 Simple State Transition FSM GL

Here, the mutations are explained, and the results obtained are shown in *Table 4.2*.

1)Mutation m0:

U10 is a wire in the design.

In this mutation, we added a D-FF to the U10 wire for Simple State Transition FSM GL design.

2) Mutation m1:

U7 is a wire in the design.

In this mutation, we added a D-FF to the U7 for Simple State Transition FSM GL design.

3) Mutation m2:

U9 is a wire in the design.

In this mutation, we added a D-FF to the U9 for Simple State Transition FSM GL design.

4) Mutation m3:

In this mutation, we removed a D-FF for the (Current State) CS [0] for Simple State Transition FSM GL design.

L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8, L_9 are the possible tests for this design passed in a single text file. The last results in *Table 4.2* are one that are remained even after all the tests are run. The Mutation m1 has no effect on the design as the outputs are same as the original design which is shown in the *Table 4.2*. The mutation is never detected in this case.

Table 4.2
Simple State Transition Gate Level FSM Mutation Results

Test	Mutation (m0 - U10)	Mutation (m1 - U7)	Mutation (m2-U9)	Mutation (m3-CS [0])
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	0	0	0	0
L_2	0	0	0	0
L_3	0	0	1	0
L_4	0	0	0	0
L_5	1	0	0	1
L_6	0	0	0	1
L_7	1	0	1	1
L_8	0	0	0	1
L_9	0	0	0	0
	0	0	0	0

4.1.2 Gate Level Combinational

4.1.2.1 Large Combinational Gate Level

Here, the mutations are done for this design to be explained below and the results in the *Table 4.3*.

X [0] is least significant bit of the binary number obtained after the encoding of the inputs. C [0], C [1] and C [2] are the one-bit values of change. Similarly, D [1] and D [2] are for item dispensed. L_1 to L_57 are the possible 57 tests for this design passed in a single text file.

1)Mutation m0:

In this mutation, we added a D-FF to the X [0] that goes to the C [0] and D [2] for Large Combinational Gate Level design.

2) Mutation m1:

In this mutation, we added a D-FF to the X [0] that goes to the D [1] for Large Combinational Gate Level design.

3) Mutation m2:

In this mutation, we added a D-FF to the X [0] that goes to the C [1] and D [1] for Large Combinational Gate Level design.

4) Mutation m3:

In this mutation, we added a D-FF to the X [0] that goes to the C [2] and D [1] for Large Combinational Gate Level design.

Table 4.3
Large Combinational Gate Level Mutation Results

Test	Mutation (m0 – X [0] for C [0] & D [2])	Mutation (m1 – X [0] for D [1])	Mutation (m2 – X [0] for C [1] & D [1])	Mutation (m3 – X [0] for C [2] & D [1])
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	0	0	0	0
L_2	0	0	0	0
L_3	0	0	0	0
L_4	0	0	0	0
L_5	0	0	0	0
L_6	0	0	0	0
L_7	0	0	0	0
L_8	0	0	0	0
L_9	0	0	0	0
L_10	0	0	0	0
L_11	0	0	0	0
L_12	0	0	0	0
L_13	1	0	0	0
L_14	1	0	0	0
L_15	0	0	0	0
L_16	0	0	0	0
L_17	1	0	0	0
L_18	1	0	0	0
L_19	0	1	1	1
L_20	0	1	1	1
L_21	0	0	0	0

Test	Mutation (m0 – X [0] for C [0] & D [2])	Mutation (m1 – X [0] for D [1])	Mutation (m2 – X [0] for C [1] & D [1])	Mutation (m3 – X [0] for C [2] & D [1])
	Detected at end	Detected at end	Detected at end	Detected at end
L_22	0	0	0	0
L_23	0	0	0	0
L_24	0	0	0	0
L_25	0	0	1	0
L_26	0	0	1	0
L_27	0	0	1	0
L_28	0	0	1	0
L_29	1	0	0	0
L_30	1	0	0	0
L_31	0	0	1	0
L_32	0	0	1	0
L_33	1	0	1	0
L_34	1	0	1	0
L_35	1	0	1	0
L_36	1	0	1	0
L_37	1	0	0	1
L_38	1	0	0	1
L_39	0	0	0	0
L_40	0	0	0	0
L_41	0	0	0	1
L_42	0	0	0	1
L_43	0	0	0	1
L_44	0	0	0	1
L_45	1	0	0	0
L_46	1	0	0	0
L_47	0	0	0	0
L_48	0	0	0	0
L_49	1	0	0	0
L_50	1	0	0	0
L_51	1	0	1	0
L_52	1	0	1	0
L_53	1	0	1	0
L_54	1	0	1	0
L_55	1	1	1	1

Test	Mutation (m0 – X [0] for C [0] & D [2])	Mutation (m1 – X [0] for D [1])	Mutation (m2 – X [0] for C [1] & D [1])	Mutation (m3 – X [0] for C [2] & D [1])
	Detected at end	Detected at end	Detected at end	Detected at end
L_56	1	1	1	1
L_57	X	X	X	x

4.1.2.2 Simple Combinational Gate Level

Here, how the mutations were done for this design will be explained below with the results in *Table 4.4*.

x_in is the input to the design.

1)Mutation m0:

In this mutation, we added a D-FF to the x_in for (Next State) NS [1] while anding with (Current State) CS [1] for Simple Combinational Gate Level design.

2) Mutation m1:

In this mutation, we added a D-FF to the x_in for Simple Combinational Gate Level design.

3) Mutation m2:

In this mutation, we added a D-FF to the x_in for one in NS [1] while ANDing with CS [1] and in NS [0] while ANDing with CS [0] for Simple Combinational Gate Level design.

4) Mutation m3:

In this mutation, we added a D-FF to the x_in for all of NS [1] for Simple Combinational Gate Level design.

Test Line says the line in the text file which we checked for eight possible cases.

Table 4.4
Simple Combinational Gate Level Mutational Results

Test Line	Mutation (m0 – NS [1] in x_in)	Mutation (m1-all x_in)	Mutation (m2-NS [1] & NS [0] for x_in)	Mutation (m3-NS [1] x_in)
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	0	0	0	1
L_2	0	0	0	1
L_3	0	0	0	1
L_4	0	0	1	1
L_5	0	1	0	1
L_6	0	1	0	1
L_7	0	1	1	1
L_8	0	1	1	1

4.2 Behavioral Level

Here, the designs are made in the behavioral level code using model-sim in system Verilog.

4.2.1 Behavioral Level FSM

4.2.1.1 Simple State Transition Behavioral Level FSM

Here, how the mutations are done for this design will be explained below with the results in *Table 4.5*.

x_in is the input to the design. S0 is state zero, S1 is state one and S3 is state three.

1)Mutation m0:

In this mutation, we added a D-FF to the x_in of S0 for Simple State Transition Behavioral Level FSM design.

2) Mutation m1:

In this mutation, we added a D-FF to the x_in of S1 for Simple State Transition Behavioral Level FSM design.

3) Mutation m2:

In this mutation, we added a D-FF to the ~x_in of S3 for Simple State Transition Behavioral Level FSM design.

4) Mutation m3:

In this mutation, we removed a D-FF from the (Current State) CS [0] for Simple State Transition Behavioral Level FSM design.

The test text file has nine lines of possible cases as shown in *Table 4.5*. From the *Table 4.5*, it can be clearly seen that mutation m2 has never been detected.

Table 4.5.
Simple State Transition Behavioral Level FSM Mutation Results

Test	Mutation (m0 - so-x_in)	Mutation (m1-s1 - x_in)	Mutation (m2-s3-~x_in)	Mutation (m3-CS [0])
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	0	0	0	0
L_2	1	0	0	1
L_3	0	0	0	0
L_4	0	0	0	0
L_5	1	1	0	0
L_6	0	1	0	0
L_7	1	1	0	0
L_8	0	1	0	0
L_9	1	1	0	1
	1	1	0	1

4.2.1.2 Elevator Behavioral Level FSM

Here, how the mutations are done for this design will be explained below with the results in *Table 4.6*.

S0, S1, and S2 are states in the design.

1)Mutation m0:

In this mutation, we added a D-FF to the up of S0 for Elevator Behavioral Level FSM design.

2) Mutation m1:

In this mutation, we added a D-FF to the down of S1 for Elevator Behavioral Level FSM design.

3) Mutation m2:

In this mutation, we added a D-FF to the down of S2 for Elevator Behavioral Level FSM design.

4) Mutation m3:

In this mutation, we removed a D-FF from the (Current State) CS [0] for Elevator Behavioral Level FSM design.

As shown in the *Table 4.6*, the test text file has been made with 10 possible cases with the test as shown. From the *Table 4.6*, it has been clearly seen that mutation m1 has never been detected.

Table 4.6
Elevator Behavioral Level FSM Mutation Results

Test Line	Mutation (m0 - up s0)	Mutation (m1 -down s1)	Mutation (m2- down s2)	Mutation (m3-CS [0])
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	1	0	0	1
L_2	1	0	0	1
L_3	1	0	0	1
L_4	1	0	0	1
L_5	0	0	1	1
L_6	0	0	0	0
L_7	0	0	0	0
L_8	1	0	0	1
L_9	1	0	0	1
L_10	1	0	0	1
	0	0	0	0

4.2.2 Behavioral Level Combinational

4.2.2.1 Simple Combinational Behavioral Level

Here, how the mutations are done for this design will be explained below with the results in *Table 4.7*.

x_in is the input to the design. S0 and S2 are states.

1)Mutation m0:

In this mutation, we added a D-FF to the x_in of S0 for Simple Combinational Behavioral Level design.

2) Mutation m1:

In this mutation, we added a D-FF to the x_in of S2 for Simple Combinational Behavioral Level design.

3) Mutation m2:

In this mutation, we added a D-FF to the x_in only and not ~x_in for Simple Combinational Behavioral Level design.

4) Mutation m3:

In this mutation, we added a D-FF to the x_in main input that goes for both x_in and ~x_in for Simple Combinational Behavioral Level design.

The test text file was made with nine possible cases and all the possibilities in the file are represented the test vector column of the *Table 4.7*.

Table 4.7
Simple Combinational Behavioral Level Mutation Results

Test Line	Mutation (m0 - so in x_in)	Mutation (m1-s2 in x_in)	Mutation (m2-x_in)	Mutation (m3-both x_in and ~x_in)
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	0	0	0	0
L_2	1	0	1	1
L_3	1	0	1	1
L_4	0	0	1	1
L_5	0	0	1	1
L_6	0	0	1	1
L_7	0	1	1	1
L_8	0	0	1	1
L_9	0	0	1	1
	0	0	0	0

4.2.2.2 Combinational Behavioral Level_1

Here, how the mutations are done for this design will be explained below with the results in *Table 4.8*.

up and down are inputs to the design. S0, S1, and S2 are states.

1)Mutation m0:

In this mutation, we added a D-FF to the up of S0 of Combinational Behavioral Level_1 design.

2) Mutation m1:

In this mutation, we added a D-FF to the up of S1 of Combinational Behavioral Level_1 design.

3) Mutation m2:

In this mutation, we added a D-FF to the up of S0 and S1 of Combinational Behavioral Level_1 design.

4) Mutation m3:

In this mutation, we added a D-FF to the down in S1 and S2 of Combinational Behavioral Level_1 design.

The test text file has been made with five possible cases.

From the *Table 4.8*, it has been clearly seen that mutation m3 has been never detected.

Table 4.8
Combinational Behavioral Level_1 Mutation Results

Test Line	Mutation (m0 - up S0)	Mutation (m1- up S1)	Mutation (m2 - S0 & S1 up)	Mutation (m3-S1&S2 down)
	Detected at end	Detected at end	Detected at end	Detected at end
L_1	0	0	0	0
L_2	1	0	1	0
L_3	0	0	0	0
L_4	0	1	1	0
L_5	0	0	0	0

4.2.2.3 Combinational Behavioral Level_2

Here, how the mutations are done for this design will be explained below with the results in *Table 4.9*.

Q is the input quarter. A1, A2, A3, A4, A5, and A6 are the items that can be selected.

1)Mutation m0:

In this mutation, we added a D-FF to the Q of Combinational Behavioral Level_2 design.

2) Mutation m1:

In this mutation, we added a D-FF to the A1 of Combinational Behavioral Level_2 design.

3) Mutation m2:

In this mutation, we added a D-FF to the A2 of Combinational Behavioral Level_2 design.

5)Mutation m3:

In this mutation, we added a D-FF to the A3 of Combinational Behavioral Level_2 design.

6) Mutation m4:

In this mutation, we added a D-FF to the A4 of Combinational Behavioral Level_2 design.

3) Mutation m5:

In this mutation, we added a D-FF to the A5 of Combinational Behavioral Level_2 design.

4) Mutation m6:

In this mutation, we added a D-FF to the A6 of Combinational Behavioral Level_2 design.

The test text file has been passed with possible tests (L_1 to L_37) and checked at the times as shown in the *Table 4.9*. The values for the respective mutations are the XOR values of output bits of original design and mutated design. The values are C0, C1, C2, D0, D1, and D2 respectively.

Table 4.9
Combinational Behavioral Level_2 Mutation Results

XOR Values (C[0], C[1], C[2], D[0], D[1], D[2])							
Test	Q_mutation	Mutation_A1	Mutation_A2	Mutation_A3	Mutation_A4	Mutation_A5	Mutation_A6
L_1	x,x,x,x,x,x	x,x,x,x,x,x	x,x,x,x,x,x	x,x,x,x,x,x	x,x,x,x,x,x	x,x,x,x,x,x	x,x,x,x,x,x
L_2	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_3	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_4	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_5	1,1,1,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_6	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_7	1,0,1,0,1,0	1,0,0,1,0,0	0,0,1,1,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_8	1,0,0,0,1,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_9	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_10	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_11	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_12	1,1,1,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_13	0,1,0,1,1,0	0,0,0,0,0,0	0,0,0,0,0,0	1,0,1,0,1,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_14	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,1,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_15	0,1,0,1,1,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0

XOR Values (C[0], C[1], C[2], D[0], D[1], D[2])							
Test	Q_mutation	Mutation_A1	Mutation_A2	Mutation_A3	Mutation_A4	Mutation_A5	Mutation_A6
L_16	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_17	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_18	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_19	1,1,0,0,0,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,1,0,1,1,0	0,0,0,0,0,0	0,0,0,0,0,0
L_20	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,1,0,1,1,0	0,0,0,0,0,0	0,0,0,0,0,0
L_21	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,1,0,1,1,0	0,0,0,0,0,0	0,0,0,0,0,0
L_22	1,1,0,0,0,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_23	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_24	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_25	0,0,0,1,0,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	1,1,0,0,0,1	0,0,0,0,0,0
L_26	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,1	0,0,0,0,0,0
L_2	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,1	0,0,0,0,0,0

XOR Values (C[0], C[1], C[2], D[0], D[1], D[2])							
Test	Q_mutation	Mutation_A1	Mutation_A2	Mutation_A3	Mutation_A4	Mutation_A5	Mutation_A6
7							
L_28	1,1,1,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,1	0,0,0,0,0,0
L_29	0,0,1,1,0,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_30	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_31	1,0,0,0,1,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,1,0,1
L_32	1,1,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,1,0,1
L_33	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,1,0,1
L_34	1,1,1,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,1,0,1
L_35	1,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,1,0,1
L_36	1,0,1,0,1,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0
L_37	1,0,1,0,1,1	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0	0,0,0,0,0,0

CHAPTER 5: CONCLUSION AND FUTURE WORK

Conclusion

This research brings a new dimension of time into mutation testing. Currently, only the functional operators, such as, arithmetic, logical and relational operators are being used in mutation testing. A significant number of design bugs are related to signal timing, before this research there was no way to do mutation testing for these signal timing bugs. This research expands the mutation testing into the time domain with use of time-shift operator. The research shows that the time-shift operator can be successfully used for mutation testing on behavioral-combinational, behavioral-sequential, gate-level-combinational, and gate-level-sequential designs. This research has used the D-FF as the time-shift operator. For sequential designs this research shows that signal time can be speeded up or slowed down by removing and adding D-FF respectively. For pure combinational designs, D-FF can only be added, as no memory elements are present in the Golden design, so signal timing can be slowed down but not speedup. This research tested each design with multiple mutations. Each mutation was individually inserted and all the tests were applied to get the results.

The results show that mutation testing using the time-shift operator is possible. Time-shift operator can find cases where certain signal with timing fault cannot be detected.

Future Work

Below are some possible research and implementation ideas for future work.

Further research should be conducted to test which other time-shift operators would be suitable for mutation testing, other than the D-Flip Flop used in this research.

The time-shift operator was inserted randomly in the design for mutation testing. It would be desirable to develop an algorithm to identify insertion location for the time-shift operator more intelligently. This intelligent algorithm would be especially useful in case of large designs.

Once the random location was identified, the time-shift operator was manually inserted in the design. This procedure is error-prone and time consuming. A better way would be to automate the insertion process, once the location has been identified either manually of using a intelligent algorithm.

The designs chosen for testing time-shift operator was not guided by any methodology. It would be more appropriate if a set of standardize or well published designs were used. This would increase the confidence that time-shift operator can be universally applied to any design.

REFERENCES

[1] P. Lisherness, N. Lesperance, and K.-T. Cheng, "Mutation analysis with coverage discounting," pp. 31-34: EDA Consortium.

[2] T. Xie, W. Mueller, and F. Letombe, "Mutation-analysis drove functional verification of a soft microprocessor," pp. 283-288: IEEE.

[3] N. Bombieri, F. Fummi, G. Pravadelli, M. Hampton, and F. Letombe, "Functional qualification of TLM verification," pp. 190-195: European Design and Automation Association.

[4] K. Huang, P. Zhu, R. J. Yan, and X. L. Yan, "Functional Testbench Qualification by Mutation Analysis," *VLSI DESIGN*, vol. 2015, pp. 1-9, 2015.

[5] Y. Serrestou, V. Beroulle, and C. Robach, "Functional Verification of RTL Designs driven by Mutation Testing metrics," pp. 222-227: IEEE.

[6] H.-Y. Lin *et al.*, "A probabilistic analysis method for functional qualification under mutation analysis," presented at the Proceedings of the Conference on Design, Automation, and Test in Europe, Dresden, Germany, 2012.

[7] https://archive.cnx.org/contents/27336337-8b30-455f-8620-a9547afede27@1/chapter-5-dsd-moore-and-mealy-state-machines

[8] https://archive.cnx.org/contents/27336337-8b30-455f-8620-a9547afede27@1/chapter-5-dsd-moore-and-mealy-state-machines

[9] http://www.32x8.com/var7.html

[10] http://www.asic-world.com/verilog/pli.html

[11] https://stackoverflow.com/questions/8964225/verilog-compilation-error-unexpected-expecting-identifier-or-type-ident

[12] https://www.doulos.com/knowhow/sysverilog/tutorial/constraints/

[13] https://asic4u.wordpress.com/category/system-verilog/

[14] http://www.asicguru.com/system-verilog/tutorial/sv-arrays/4/

[15] http://verilog.renerta.com/source/vrg00013.htm

[16] http://testbench.in/SV_09_ARRAYS.html

[17] T. B. Nguyen, C. Robach,"Mutation Testing Applied to Hardware: The Mutants Generation", LCIS-ESISAR, BP 54, 50 rue B. de Laffemas, 26902, Valence, France.

[18] N. Bombieri, F. Fummi, V. Guarnieri, and G. Pravadelli, "Testbench Qualification of SystemC TLM Protocols through Mutation Analysis," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1248-1261, 2014.

[19] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Application of genetically engineered finite-state-machine sequences to sequential circuit ATPG," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 3, pp. 239-254, 1998.

[20] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649-678, 2011.

[21] Y. Serrestou, V. Beroulle, and C. Robach, "Impact of hardware emulation on the verification quality improvement," pp. 218-223: IEEE.

[22] Alsubaei, S., S. M. Qaisar, and W. Alhalabi. 2017. "A VHDL based Moore and Mealy FSM example for education." In *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)*, 456-59.

[23] https://courses.cs.washington.edu/courses/csep567/10wi/lectures/Lecture2.pdf

[24] https://inst.eecs.berkeley.edu/~cs150/Documents/FSM.pdf

APPENDIX A: VENDING MACHINE TRANSITIONS

Table A.1
Change Assignment

Change	Assignment ($C_2C_1C_0$)
0	000
0.25	001
0.50	010
0.75	011
1.00	100
1.25	101
1.50	110

Above *TableA.1*, depicts how the change output is assigned to 3 bits as $C_2C_1C_0$.

Table A.2
Input Encoding

Input Encoding ($X_3X_2X_1X_0$)	
Dime	000
A1	001
A2	010
A3	011
A4	100
A5	101
A6	110
Return Change	111

The above *TableA.2,* depicts how the 8 – different inputs are encoded into 4 inputs X3X2X1X0.

Table A.3
Output Encoding

Output Encoding(DC)		
0.00 0.25 0.50 0.75 1.00 1.25 1.50	C0 C1 C2	C
Dispense A1 Dispense A2 Dispense A3 Dispense A4 Dispense A5 Dispense A6	D0D1D2	D

The above *Table A.3*, explains how the outputs are obtained.

Table A.4
State Transition Table

State Transition Table ($Q^+_2Q^+_1Q^+_0$)								
State	Inputs ($X_3X_2X_1X_0$)							
(Q2Q1Q0)	1000	1001	1010	1011	1100	1101	1110	1111
0 0 0	0 0 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	0 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 1 0	0 1 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 1 1	1 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
1 0 0	1 0 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
1 0 1	1 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
1 1 0	1 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0

The above *TableA.4*, depicts the transition of states depending on the inputs.

Table A.5
Change Table

Change Table ($C_2C_1C_0$)								
State	Inputs ($X_3X_2X_1X_0$)							
(Q2Q1Q0)	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	0 0 0	0 0 0	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
0 1 0	0 0 0	0 0 1	0 0 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
0 1 1	0 0 0	0 1 0	0 0 1	0 0 0	0 1 1	0 1 1	0 1 1	0 1 1
1 0 0	0 0 0	0 1 1	0 1 0	0 0 1	0 0 0	1 0 0	1 0 0	1 0 0
1 0 1	0 0 0	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0	1 0 1	1 0 1
1 1 0	0 0 1	1 0 1	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0	1 0 0

The above *Table A.5* depicts the change obtained when an input is chosen in that state.

Table A.6
Dispense Table

Dispense Table ($D_2D_1D_0$)								
State	Inputs (X_3 $X_2X_1X_0$)							
(Q2Q1Q0)	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	0 0 0	0 0 1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 1 0	0 0 0	0 0 1	0 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 1 1	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0	0 0 0	0 0 0	0 0 0
1 0 0	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	0 0 0	0 0 0	0 0 0
1 0 1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	0 0 0	0 0 0
1 1 0	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	0 0 0

The above *TableA.6,* depicts the items dispensed upon selection of inputs in that state.