A Project Report on

FAULT TESTING OF REVERSIBLE CIRCUIT

Project Report submitted in partial fulfilment of the requirements for the 6th Semester of Master of Computer Application (MCA) under University of Calcutta

: SUBMITTED BY:

SUDESHNA HALDAR	AKASH PATRA
Roll No: 91/MCA/190019	Roll No: 91/MCA/190015
Registration No: D01-1211-0048-19	Registration number: 432-1121-0869-15

MCA 6TH SEMESTER

Under the supervision of

Joyati Mondal,

Assistant Professor

A.K. Choudhury School of

Information Technology

UNIVERSITY OF CALCUTTA

A.K. CHOUDHURY SCHOOL OF INFORMATION TECHNOLOGY
UNIVERSITY OF CALCUTTA

University of Calcutta, JD-2 Sector-III, Saltlake, Kolkata-700106, INDIA

CERTIFICATE

This is to certify that the project synopsis entitled **"Fault Testing of Reversible Circuit"** submitted for partial fulfilment of the requirements of 6th Semester of Master of Computer Application (MCA) under University Of Calcutta; has been carried out by **Akash Patra** (**Roll No- 91/MCA/190015** and **Registration No-432-1121-0869-15**) and **Sudeshna Haldar** (**Roll No- 91/MCA/190019** and **Registration No- D01-1211-0048-19**) under the supervision of **Joyati Mondal, Assistant Professor of A.K. Choudhury School of Information Technology UNIVERSITY OF CALCUTTA .**

CHAIRMAN, BOARD OF EXAMINER

Asst.Prof. Joyati Mondal,
AKCSIT, University of Calcutta,
Project Supervisor

External Examiner(s)

ACKNOWLEDGEMENT

I wish to express my profound sense of gratitude of my project supervisor Joyati Mondal Assistant Professor of AKCSIT, University of Calcutta, for his support, inspiration and guidance. He has showed me different ways to approach a problem. I have also learned from him that an approach needs to be persistent to accomplish my goal. I am immensely grateful to him for giving his valuable time and constant advice for discussing various ideas related to my project work it is being precious learning experience for me to work under tutelage.

I am also thankful to my department; A.K. Choudhury School of I.T, University of Calcutta; for providing me with the required resources for working on this project.

Lastly I like to express my heartiest gratitude to my seniors and my friends; and to all who have directly or indirectly extended their valuable guidance and advice during the preparation of this project; which will give me the continuous flow of inspiration to complete the project.

THANK YOU

Date: -----------------------------

Akash Patra

Sudeshna Haldar

Table of Contents

ABSTRUCT: 1

PRELIMINARIES: 1

- Reversible Logic Function: 1
- Reversible Logic Gates: 1
  - NOT GATE: 2
  - FEYNMAN GATE 2
  - TOFFOLI GATE: 3
  - Multiple Control Toffoli (MCT) Gate: 3
  - FREDKIN GATE: 3
  - PERES GATE: 4
- Reversible library: 4
  - NCT library: 5
- What is a fault? 6
- Fault Model: 6
  - i)Stuck At Fault: 7
  - iii)Missing Gate Fault: 7
  - iv) Cross Point Fault: 9
  - v) Cell Fault: 9

OBJECTIVE: 9

- TESTING: 9
- FAULT IDENTIFICATION OF DIFFERENT GATES: 10
  - Stuck-at Fault Identification 10
  - Bridging Fault Identification 10
  - Cross Point Fault Identification 10
  - Cell Fault Identification 10
  - Missing Gate Fault Identification 11

Implementation: 12

- Project structure 12
- Utilities 12
  - BitTransform: 13
  - BitTransformException: 13
  - Code: 13
- Implementation of circuit and test set: 14
  - Circuit: 14
  - Test Set: 24

MISSING GATE FAULT DETECTION AND TESTING:........ 26
Single Missing Gate Fault (SMGF):........ 27
Partial Missing Gate Fault (PMGF):........ 28
Coding for Missing Gate Fault (SMGF & PMGF) : 33
Package for MGF model : 33
Package for SMGF fault model : 38
Package for PMGF fault model : 41
Graphics User Interface 46
Package for GUI 46
Classes which hold each individual window of our application 50
Screenshot of our application: 67
Overall execution:........ 67
Validation: 69
SUMMARY: 74
FUTURE WORK:........ 74
REFERENCES: 75

ABSTRUCT:

Irreversible computation is where two or more bits are combined to produce one output for example AND gate, OR gate, NOT gate etc. This is known as bit erasure in which data is lost which is transformed into heat. To minimize this energy dissipation C.H.Bennette referred to use reversible circuit.

Reversible circuit has two properties:

1) The no. of input will be equal to the no. of outputs
2) The logic of the reversible circuit will be a bijective function i.e, a particular input will produce a distinct output.

Different faults can occur in the manufacturing process of reversible circuit. In order to find these faults or errors we have to test them. This paper illustrates different methods to find a proper test set which can identify different errors in the circuit.

PRELIMINARIES:

Reversible Logic Function:

A logic function $f(x_1, x_2, \ldots, x_n)$ of n Boolean variables is called a reversible function if it realizes bijective functions, i.e., $f: B^n \Rightarrow B^m$. The property of the bijective function is the one-to-one property and onto property. The logic function is said bijective function because-

in reversible computation each possible input pattern has a unique output pattern(one-to-one).

For example,

Input		$f(x_1, x_2) = (x_1, x_1 \oplus x_2)$	
x_1	x_2	x_1'	x_2'
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Reversible Logic Gates:

Reversible logic gates are formed by using reversible logic function. Generally, these logic gates are denoted as-$g(C, T)$ where C is the set of Control lines and T is the set of Target lines. In these gates control line and the target line should be different. If there cannot have any control line but it must have at least one target line, for example NOT gate.

NOT GATE:

A NOT gate is the same as classical not gate. It is called an inverter. A NOT gate performs logical negation on its input. In other words, if the input is 0, then the output will be 1. Similarly, a 1 input results in a 0 output.

We are considered this is as reversible logic gate because-

a) Input can be easily inferred from the output

b) Each input produces distinct output.

INPUT	OUTPUT
0	1
1	0

Figure 1 Truth Table for NOT Gate

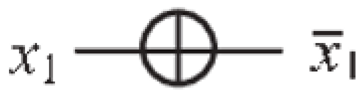


Figure 2 Circuit Diagram of NOT Gate

FEYNMAN GATE:

It can perform negation operation but in controlled way. So, this reversible gate is also called Controlled NOT (CNOT) Gate. If two inputs are A and B, the first input A is known as CONTROL line and second input B is known as TARGET line. Operation on target line is negation and only performed when control line is set otherwise no operation on target line.

When A = 0 then Q = B, when A = 1 then Q = B'.

INPUT		OUTPUT	
A	B	P	Q
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Figure 3 Truth Table of FEYNMAN Gate

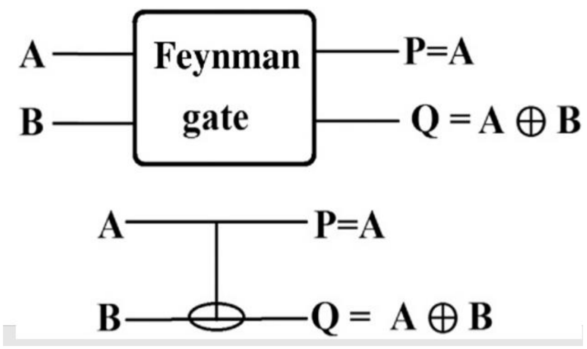


Figure 4 Circuit Diagram of FEYNMAN Gate

TOFFOLI GATE:

This gate is known as 3×3 gate and can be generalized up to n×n size. As per definition target line flips when all control lines are set. The Toffoli gate passes the first two inputs through and inverts the third if the first two are both 1.

INPUT			OUTPUT		
A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figure 5 Truth Table of TFFOLI Gate

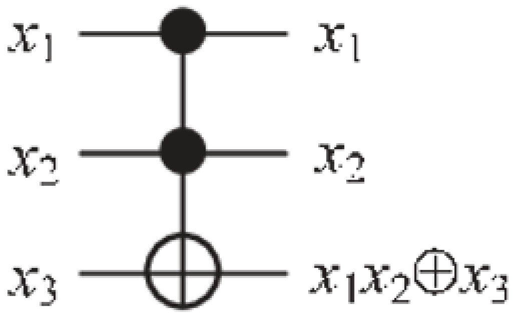


Figure 6 Circuit Diagram of TOFFOLI Gate

Multiple Control Toffoli (MCT) Gate:

It is the generalised version of Toffoli gate. The equation is f(k_m, T) = k_1.k_2…k_m⨁T.
Here, Output is inverted when all the inputs are 1.

Input					Output				
k_1	k_2	…	k_m	T	k_1'	k_2'	…	k_m'	f
0	0	…	0	0	0	0	…	0	0
0	0	…	0	1	0	0	…	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	…	1	1	1	1	1	1	0

Figure 7 Truth Table of MCT Gate

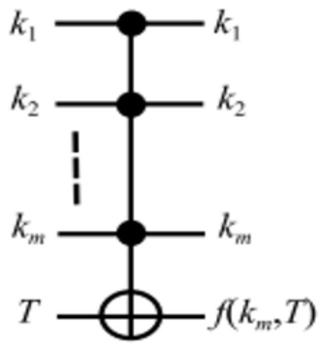


Figure 8 Circuit Diagram of MCT Gate

FREDKIN GATE:

Fredkin Gate is a fundamental concept in reversible and quantum computing reversible and quantum computing. Every Boolean function can be built from 3 * 3 Fredkin gates:
P = A,
Q = if A then C else B,
R = if A then B else C.

INPUT			OUTPUT		
A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

Figure 9 Circuit Diagram of FREDKIN Gate

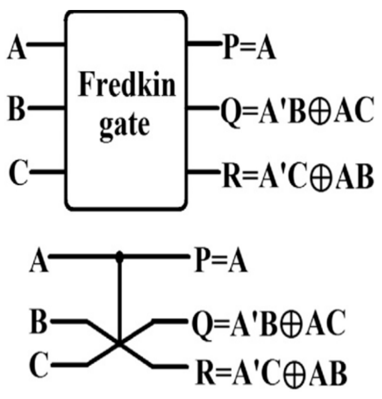


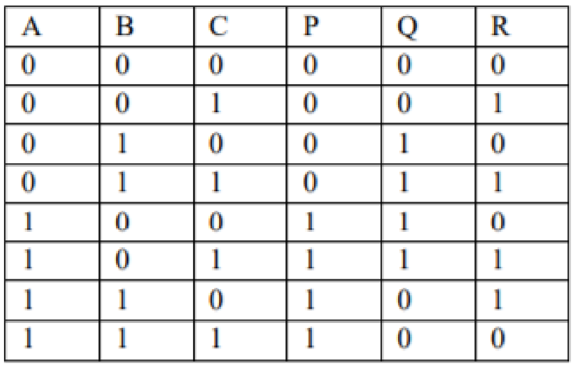
Figure 10 Circuit Diagram of FREDKIN Gate

It is also called a Swap gate because the target input is swapped whenever all control inputs are at logic 1.

If we extend this gate to more than one control input then it is called Multiple Control Fredkin (MCF) gate.

PERES GATE:

Peres gate which is a 3*3 gate having inputs (A, B, C) and outputs P = A; Q = A XOR B; R=AB XOR C. It has Quantum cost four.



A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Figure 11 Truth Table for PERES Gate

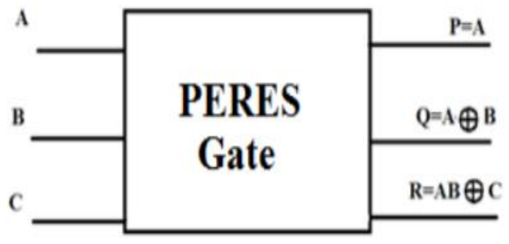


Figure 12 Circuit Diagram of PERES GATE

Reversible library:

An n library is the set of n ×n reversible gates which are used to synthesize n ×n reversible gates, denoted as n_L, or simply as L.

We can use T (L) to denote a set of all n × n reversible gates that can be synthesized using gates from library L.

A universal library L satisfies that all n × n reversible gates can be synthesized by L.

A minimal universal library L is a universal library such that there does not exist a universal library L_0 such that |L₀| < |L.

NCT library (NOT gate, CNOT gate, Toffoli gate.) is one of the most popular reversible library which is used in many fields of reversible computation.

NCT library:

The C^nNOT gate is used as the main reversible gate to build any reversible circuit, since it is proved to be universal for reversible logic synthesis.

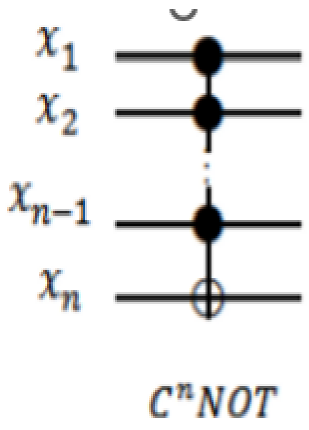


Figure 13

The action of C^nNOT gate is defined as follows, if the control bit lines are set to 1 then the target bit line is flipped, otherwise the target bit line is left unchanged. Some special cases of the C^nNOT gate are defined as follows, C^1NOT gate with no control bit is called NOT gate. C^2NOT with one control bit is called $CNOT$ gate. C^3NOT with two control bits is called Toffoli gate. For the sake of readability C^1NOT, C^2NOT and C^3NOT will be written shortly as N, C and T respectively where the control and/or target bits will be shown in the subscript of the gate and the total number of bits will be shown in the superscript. The N, C and T gates can be used to form a universal library for 3-in/out reversible circuits known as NCT library. The main NCT library consists of 12 gates.

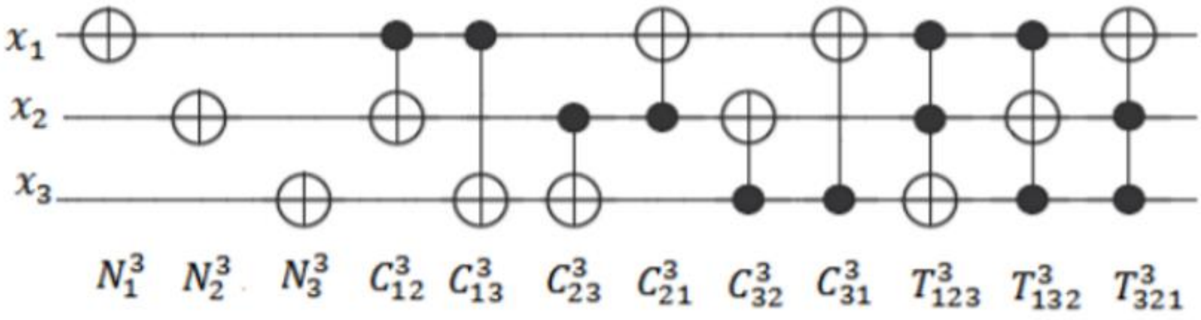

x_1
x_2
x_3
N_1^3 N_2^3 N_3^3 C_{12}^3 C_{13}^3 C_{23}^3 C_{21}^3 C_{32}^3 C_{31}^3 T_{123}^3 T_{132}^3 T_{321}^3

Figure 14 The main NCT library consists 12 gates

The other types of reversible libraries are:

NCTF Library

Consist of NOT, CNOT, TOFFOLI, FREDKIN gate.

NCPT Library

Consist of NOT, CNOT, TOFFOLI, PERES gate.

NCP Library

Consist of NOT, CNOT, PERES gate.

NCPF Library

Consist of NOT, CNOT, PERES, FREDKIN gate.

What is a fault?

A fault is a logical model of a physical disturbance, which changes the (Boolean) function of the circuit.

In particular, a given fault model can be extended to multiple faults. For example, suppose f1, f2, f3 and f4 are faults with respect to some fault model. Then, (f1, f3) is a double fault which describes the situation where both faults are present simultaneously.

Fault Model:

There are several types of fault models. These are:

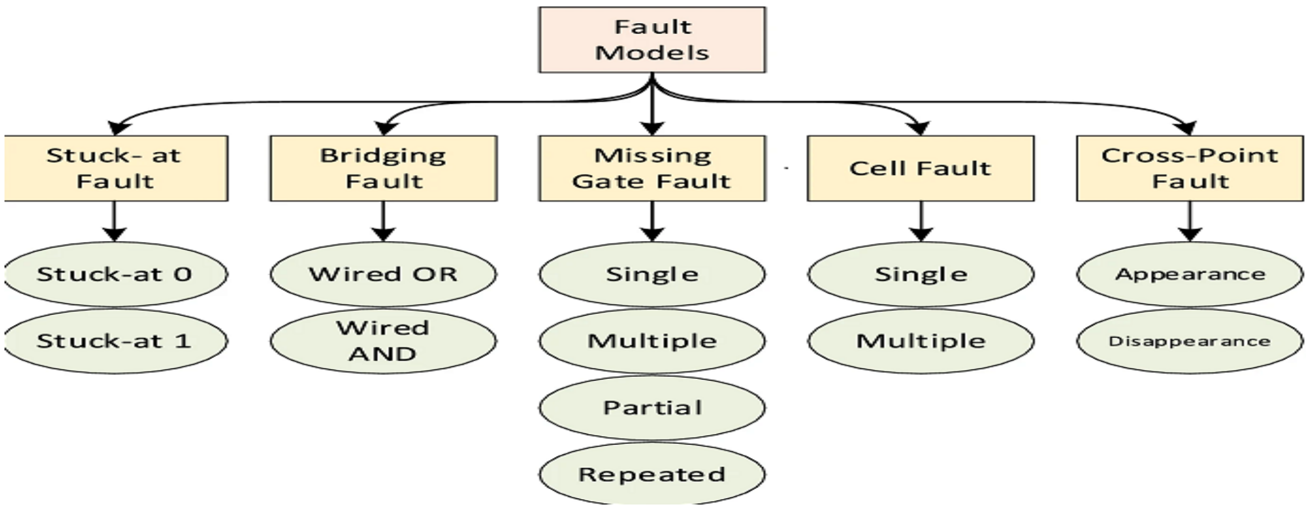


Figure 15 Different Fault Models

i)Stuck At Fault:

Stuck-at Fault is a Functional Fault on a Boolean (Logic) Function Implementation.

It is an abstract fault model. A logic stuck-at 1 means when the line is applied a logic 0, it produces a logical error. A logic error means 0 becomes 1 or vice versa. The standard stuck-at fault model used in testing conventional circuits, which includes all faults that fix the values of wires in the circuit to either 0 or 1.

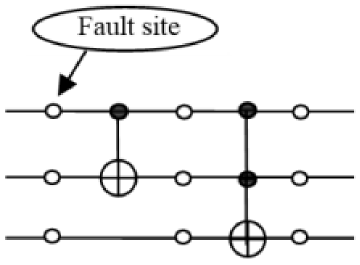


Figure 16 Stuck-at Fault Model

The faults can be aroused on single or multiple nodes at a single time which can be either of same or of both types.

In this figure stuck-at fault sites are represented by small open dots.

***ii)*Bridging Fault:**

A bridging fault consists of two signals that are connected when they should not be. Depending on the logic circuitry employed, this may result in a wired-OR or wired-AND logic function.

These faults may occur on a couple of wires or on multiple wires at respective levels of the circuit which are termed as single intra-level and multiple intra-level bridging faults.

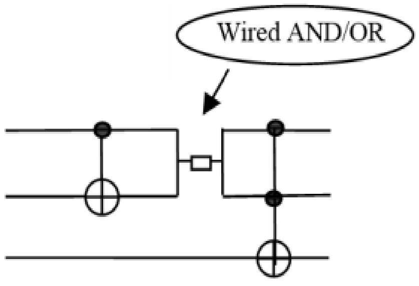


Figure 17 Bridging Fault Model

iii)Missing Gate Fault:

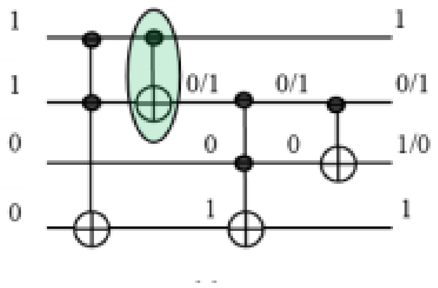


Figure 18 Missing Gate Fault Model 1

A new fault model, the missing gate fault (MGF) model, is proposed to better represent the physical failure modes of quantum technologies.

A missing gate fault is defined as complete disappearance of a reversible gate from a circuit.

This fault is of four types-a) Single Missing Gate Fault

b) Multiple Missing Gate Fault

c) Partial Missing Gate Fault

d) Repeated Gate Fault

a) Single Missing Gate Fault (SMGF):

This model corresponds to the case when one gate completely disappears from the circuit.

Maximum occurrence of SMGF in the circuit is equal to number of gates (G) contained by the circuit.

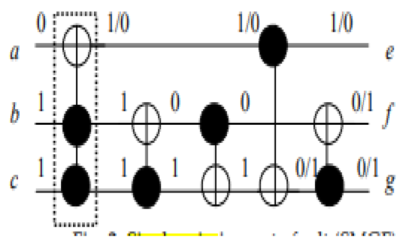


Figure 19 Single Missing Gate Fault

b) Multiple Missing Gate Fault (MMGF):

This is defined as complete disappearance of two or more consecutive gates from the circuit.

The maximum number of MMGF is given by {G(G+1)/2-G} where G is the number of gates in the circuit.

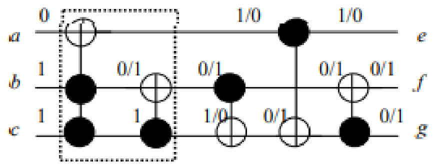


Figure 20 Multiple Missing Gate Fault

c) Partial Missing Gate Fault (PMGF):

This is used to model the defects resulting from the partially misaligned or mistuned gate pulses.

This figure shows a first-order PMGF affecting the second control input of the leftmost gate. An SMGF can be seen as a 0-order PMGF. In the circuit of this figure, if we apply {a, b, c} = {1, 0, 1}, the normal output would be {e, f, g} = {1, 1, 0}, whereas, in the presence of the first order PMGF fault as shown, the output will be {e, f, g} = {0, 0, 1}. Hence,

the vector {a, b, c} = {1, 0, 1} detects this fault. It has been shown that PMGF (of first or higher order) is detected when a 0 is applied to at least one of the affected control inputs and a 1 to all other control inputs. Thus, a higher order PMGF is detected by a test vector for a first order PMGF, the affected control input of which is one of those affected in the higher order PMGF. Hence, it is sufficient to consider first order PMGFs only.

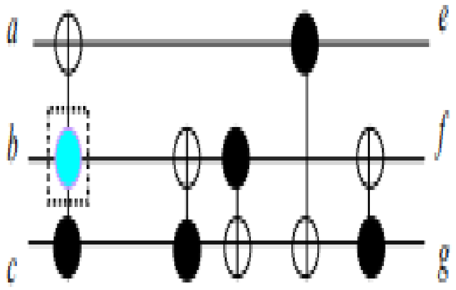


Figure 21 Partial Missing Gate Fault

d) Repeated Gate Fault (RGF):

A repeated-gate fault (RGF) is an unwanted replacement of a gate by several instances of the same gate.

The repeated-gate fault (RGF) model assumes that a gate in a circuit is repeated, i.e., shows up several times. A single occurrence of that gate is thus replaced by multiple occurrences.

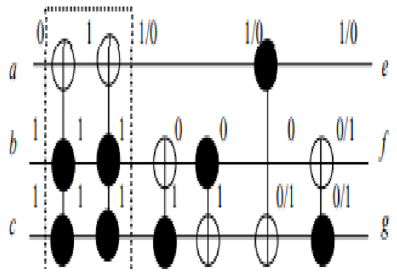


Figure 22 Repeated Missing Gate Fault

iv) Cross Point Fault:

This fault model is related to the missing or extra connection at the cross-point or control point of a gate in a reversible circuit (Zhong, & Muzio, 2009). There are two types of cross-point faults that may occur in a circuit. They are the appearance fault, which occurs when an extra control point is added on a gate, as shown in Figure 'a' and the disappearance fault, which occurs when one or more control points are missing from a gate as shown in Figure 'b'.

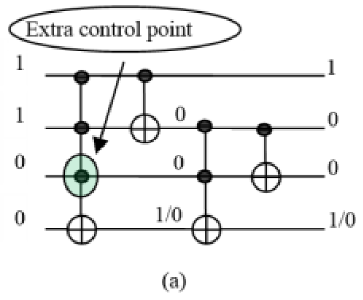


Figure 23

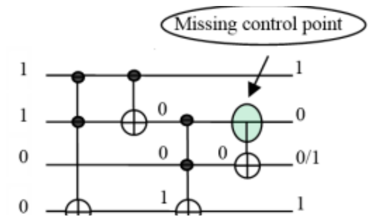


Figure 24

v) Cell Fault:

A cell fault describes the replacement of a gate's function by an arbitrary Boolean function. Here the gates are referred to as a cell. The foundation of these faults is belonging to fault modelling in cellular logic arrays and therefore these faults can be simply called by cell faults.

OBJECTIVE:

Our goal is to check whether a test set provided by the users can detect all possible faults of a particular fault model. Let's first explore the different fault models.

TESTING:

A key component in circuit design is that of testing. Testing of the reversible circuit gaining grounds since more than a decade, where the researchers have been forwarded about the kind of faults that can be happed and developing their testing strategies in this emerging field. A collective information of testing approaches from the literature is provided in this section with respect to cost metrics and test complexity. The possible categorization of testing approaches for reversible circuits that have been proposed so far are shown in the figure below. A brief about the contribution in these approaches are described in this section.

Testing can be categorized into two behavioural schemes:
(i) ***Online testing,*** where the detection of faults within the circuit is carried out during its operation
(ii) ***Offline testing,*** where test vectors are applied after extracting the circuit out from its normal operation and the correct output values are known.

It can be noted that mostly all the online testing approaches utilize parity checking technique. However, offline approaches exploit the bijective mapping property of reversible function. Number of inputs, gates, quantum cost, ancillas, and garbage are considered to evaluate the performance of respective testing approaches.

Three more measures are included for the analysis of testing approaches:

test size (s), execution time (t), and fault coverage (FC). However, test size and execution time are used in case of ATPG approaches.

FAULT IDENTIFICATION OF DIFFERENT GATES:

Among different reversible gates, k-CNOT or Multiple Control Toffoli (MCT) gate and Multiple Control Fredkin (MCF) are the most fundamental gate which are most common gates used in most of the reversible circuit.

We will essentially focus on MCT for identification of faults.

We are considering an n wire MCT gate having (k1, k2,..., km) control inputs and target input T . Respective deterministic methodologies for the identifications of faults are explained below.

Stuck-at Fault Identification

Assuming logic 0 and 1 values at all the fault sites, the n dimensional test vector of size 2 given by (0, 0,..., 0)(1, 1,..., 0) defines a complete test set for the detection of all single and multiple type stuck-at faults of an MCT gate.

For a circuit which has n input lines and m gates can have n× (m+1) no. of possible stuck-at 0 and the same no. of stuck-at 1 faults. So total no. of stuck-at faults is 2× [n×(m+1)].

Bridging Fault Identification

Bridging faults is dependent on total number of wires in the circuit. The detection is done by assuming complementary values between the two wires at every level of the circuit. The n dimensional test vector of size n, produced by shifting 0 value from first wire to next until it reaches to the end of test vector, given as (0, 1,..., 1), (1, 0,..., 1), . . . (1, 1,..., 0) is the complete test set for all single intra-level bridging faults of an MCT gate of a circuit.

Cross Point Fault Identification

The detection is achieved by assigning the combination of n test vectors keeping logic 1 to the m − 1 control inputs and 1 value to target input of each gate at distinct levels of the circuit. Assuming logic 0 to the control input where the fault has been occurred, making rest all control at logic 1 and 1 (or 0) value to target input of the gate. The n dimensional test vector of size (n − 1) given as {k1, k2,... km, T }={(0, 1,... 1, 1), (1, 0,... 1, 1), . . . (1, 1, ··· 0, 1) is the complete test set for the detection of all CPF of the gate for n ≥ 2.

Cell Fault Identification

The 2^n greedy permutation fix the detection of this type of fault and all n dimensional test vectors of size 2^n are required to detect all the cell faults in the circuit.

Missing Gate Fault Identification

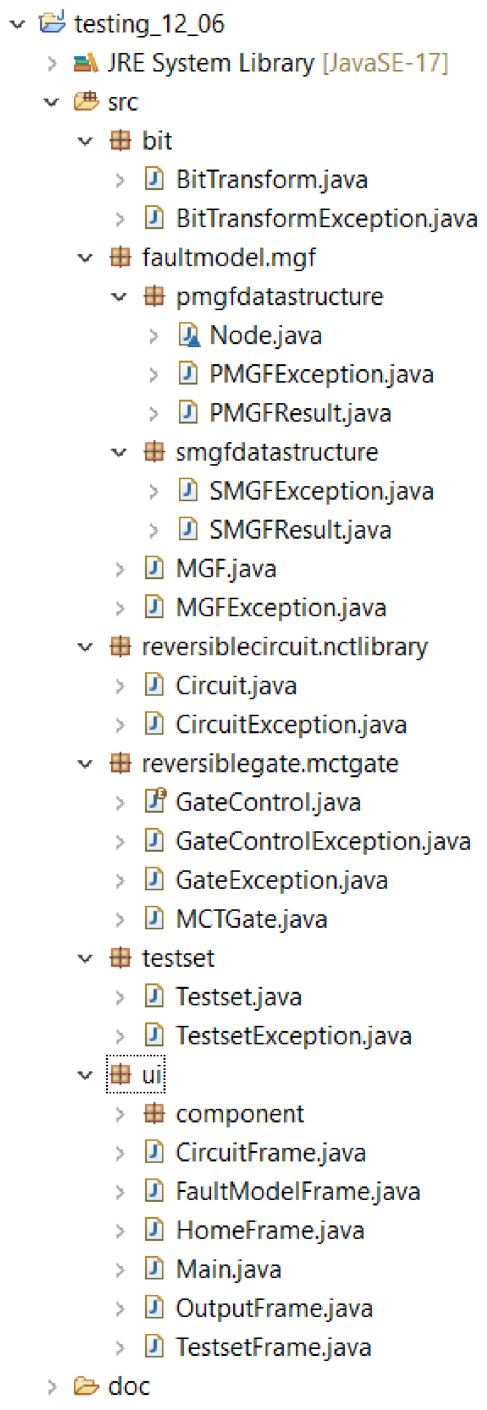
Single missing gate faults is equal to the gates present in the circuit. The detection principle for this type of fault is to assign logic values 1 to the control inputs and 1 (or 0) value to the target input of the gate in the circuit. The n-dimensional single test vector given as {k1, k2,... km, T ={(1, 1,... 1, 1)} (or {(1, 1,... 1, 0)}) is the complete test set for its detection.

Partial Missing gate fault can be detected by assigning 0 to one of the control lines of a MCT gate and assigning 1 to other control lines. This will detect one of the first order partial missing gate fault among all the possible PMGF faults of a MCT gate. So if a MCT gate has n control lines then there can be n possible PMGF faults.

We have to design a tool where user will provide a fault free circuit and a test set and our tool will check whether the test set can detect all the possible SMGF and PMGF in the circuit.

Implementation:

Project structure



```
testing_12_06
  JRE System Library [JavaSE-17]
  src
    bit
      BitTransform.java
      BitTransformException.java
    faultmodel.mgf
      pmgfdatastructure
        Node.java
        PMGFException.java
        PMGFResult.java
      smgfdatastructure
        SMGFException.java
        SMGFResult.java
      MGF.java
      MGFException.java
    reversiblecircuit.nctlibrary
      Circuit.java
      CircuitException.java
    reversiblegate.mctgate
      GateControl.java
      GateControlException.java
      GateException.java
      MCTGate.java
    testset
      Testset.java
      TestsetException.java
    ui
      component
      CircuitFrame.java
      FaultModelFrame.java
      HomeFrame.java
      Main.java
      OutputFrame.java
      TestsetFrame.java
  doc
```

Utilities

Our application will work with binary data which can be best represented by Boolean data type. On other hand users can interact with our application using integers. Thus, certain methods are required for conversion.

Package bit: This package contains classes BitTransform and BitTransformException.

BitTransform:

This class contains static methods to convert integer data to boolean type data and vice-versa.

BitTransformException:

This class represents all the possible exceptions that can occur in BitTransform class.

Code:

BitTransform.java

```
package bit;
public class BitTransform {
	public static boolean intToBool(int bit) {
		return (bit >= 1);
	}
	public static int boolToInt(boolean bit) {
		return bit ? 1 : 0;
	}
	public static int[] boolArrToIntArr(boolean[] bits) throws BitTransformException {
		if (bits == null || bits.length == 0)
			throw new BitTransformException("No array provide");
		int[] intArr = new int[bits.length];
		for (int i = 0; i < bits.length; i++) {
			intArr[i] = boolToInt(bits[i]);
		}
		return intArr;
	}
	public static boolean[] intArrToBoolArr(int[] intArr) throws BitTransformException {
		if (intArr == null || intArr.length == 0)
			throw new BitTransformException("No array provide");
		boolean[] boolArr = new boolean[intArr.length];
		for (int i = 0; i < intArr.length; i++) {
			boolArr[i] = intToBool(intArr[i]);
		}
		return boolArr;
	}
}
```

BitTransformException.java

```
package bit;

public class BitTransformException extends Exception {
	private static final long serialVersionUID = 1L;

	public BitTransformException(String message) {
		super(message);
	}
}
```

Implementation of circuit and test set:

Circuit:

For testing we take the reversible circuit as $m \times n$ matrix, where
m= number of inputs
n= number of gates

We used only MCT (Multiple Control Toffoli) gate,

Element e_{ij} for 0<=i<m and 0<=j<n can be either 0 representing “No input”, 1 representing “Control Input” or 2 representing “Target Input”.

Example:

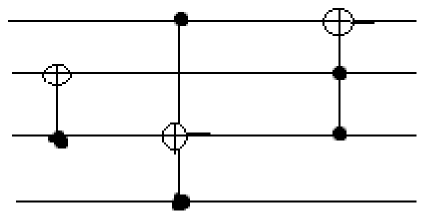


Figure 25 Actual Circuit

$$\begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 26 Matrix Representation

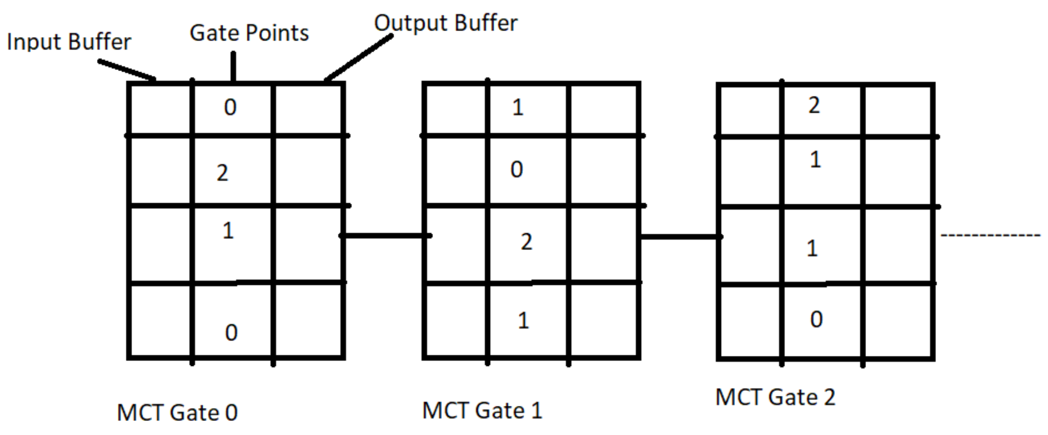


Figure 27 Implementation of the circuit

Coding for Circuit:

Package for MCT Gate

<u>GateControl.java:</u>

Class to represent individual gate points. In case of MCT Gate the possible points are – “NO_INPUT”, “CONTROL_INPUT”, “TARGET_INPUT”.

```
package reversiblegate.mctgate;

public enum GateControl {
	/**
	 * For a gate point which is not connected to the input line
	 */
	NO_INPUT(0),
	/**
	 * For a gate point which is connected to the input line as a control input
	 */
	CONTROL_INPUT(1),
	/**
	 * For a gate point which is connected to the input line as a target input
	 */
	TARGET_INPUT(2);

	private int intValue;

	private GateControl(int value) {
		intValue = value;
	}
	public int getValue() {
		return intValue;
	}
	public static GateControl intToGateControl(int control) throws GateControlException {
		switch (control) {
		case 0:
			return NO_INPUT;
		case 1:
			return CONTROL_INPUT;
		case 2:
			return TARGET_INPUT;
		}
		throw new GateControlException("Invalid control for MCT gate");
	}
	public static int gateControlToInt(GateControl control) {
		return control.getValue();
	}
}
```

GateControlException.java :

Class responsible for all the possible exception in GateControl class.

```
package reversiblegate.mctgate;
public class GateControlException extends Exception {
	private static final long serialVersionUID = 1L;

	public GateControlException(String message) {
		super(message);
	}
}
```

MCTGate.java :

Class uses GateControl class to represent individual MCT gate.

```
package reversiblegate.mctgate;

import java.util.Arrays;

/**
 * Represent individual MCT gate
 */
public class MCTGate {

	private boolean[] inputBuffer, outputBuffer;
	private final GateControl[] control;
	private final int order;

	private GateControl[] validate(int[] control) throws GateException {
		if (control == null || control.length == 0)
			throw new GateException("No control provided");
		GateControl[] newControl = new GateControl[control.length];
		int targetCount = 0;
		for (int c = 0; c < control.length; c++) {
			try {
				newControl[c] = GateControl.intToGateControl(control[c]);
			} catch (GateControlException exp) {
				GateException gExp = new GateException("Invalid Control");
				gExp.initCause(exp);
				throw gExp;
```

```
            }
            if (newControl[c] == GateControl.TARGET_INPUT)
                targetCount++;
        }
        if (targetCount != 1)
            throw new GateException("Number of target input should be exactly one");
        return newControl;
    }

    public MCTGate(int[] control) throws GateException {
        this.control = validate(control);
        this.order = this.control.length;
        this.inputBuffer = new boolean[this.order];
        this.outputBuffer = new boolean[this.order];
    }

    /**
     * Returns the number of input lines of the gate
     *
     * @return the number of input lines of the gate
     */
    public int getOrder() {
        return order;
    }

    /**
     * Returns the input buffer of the gate.
     * <p>
     * It do not returns the actual buffer but a new array containing the value of
     * the buffer.
     *
     * @return boolean array representing input buffer
     */
    public boolean[] getInputBuffer() {
        return inputBuffer.clone();
    }

    /**
     * Returns the output buffer of the gate.
     * It do not returns the actual buffer but a new array containing the value of the buffer.
     * @return boolean array representing output buffer
     */
    public boolean[] getOutputBuffer() {
        return outputBuffer.clone();
    }
```

```
/**
 * Returns a array of {@link GateControl}
 * <p>
 * Do not returns the actual array but returns a new copied array.
 *
 * @return GateControl array representing gate points
 */
public GateControl[] getControl() {
        return control.clone();
}

/**
 * Accepts a boolean array and loads the data to the input buffer of the gate.
 * <p>
 * Do not set the passed array as the input buffer, rather copy the content to
 * the buffer. Buffer is set only when the parameter is valid, otherwise throws
 * GateException.
 *
 * @param buffer boolean array from which the values will be copied to the input
 *           buffer of the gate.
 * @throws GateException thrown if the argument is null or the order of the gate
 *                 and the order of the argument mismatch
 */
public void setInputBuffer(boolean[] buffer) throws GateException {
        if (buffer == null || buffer.length != order)
                throw new GateException("Incompatible input buffer size");
        System.arraycopy(buffer, 0, inputBuffer, 0, order);
}

/**
 * Accepts a boolean array and loads the data to the output buffer of the gate.
 * <p>
 * Do not set the passed array as the output buffer, rather copy the content to
 * the buffer. Buffer is set only when the parameter is valid, otherwise throws
 * GateException.
*/
public void setOutputBuffer(boolean[] buffer) throws GateException {
        if (buffer == null || buffer.length != order)
                throw new GateException("Incompatible output buffer size");
        System.arraycopy(buffer, 0, outputBuffer, 0, order);
}

/**
 * Propagates the input through the gate and populates the output buffer with
```

```
	 * the output.
	 */
	public void propagate() {
		boolean andedControlInput = true;
		int targetIndex = -1;
		for (int i = 0; i < order; i++) {
			outputBuffer[i] = inputBuffer[i];
			if (control[i] == GateControl.TARGET_INPUT)
				targetIndex = i;
			else if (control[i] == GateControl.CONTROL_INPUT)
				andedControlInput &= inputBuffer[i];
		}
		if (andedControlInput)
			outputBuffer[targetIndex] = (!inputBuffer[targetIndex]);
	}

	/**
	 * Propagate in reverse order i.e. from output to input.
	 * <p>
	 * Compute the input from the output.
	 */
	public void reversePropagate() {
		boolean[] temp = inputBuffer;
		inputBuffer = outputBuffer;
		outputBuffer = temp;
		propagate();
		temp = inputBuffer;
		inputBuffer = outputBuffer;
		outputBuffer = temp;
	}

	/**
	 * Clears the input and output buffer.
	 * <p>
	 * Both the buffers are set to false after this operation.
	 */
	public void clearBuffers() {
		Arrays.fill(inputBuffer, false);
		Arrays.fill(outputBuffer, false);
	}
}
```

GateException.java :

Class used for any exception occurring in MCTGate class.

```
package reversiblegate.mctgate;
public class GateException extends Exception {

	private static final long serialVersionUID = 1L;
public GateException(String message) {
		super(message);
	}
}
```

Package for the Circuit using NCT library

Circuit.java :

Class for a reversible circuit

```
package reversiblecircuit.nctlibrary;

import java.util.ArrayList;
import java.util.Arrays;

import reversiblegate.mctgate.GateControl;
import reversiblegate.mctgate.GateException;
import reversiblegate.mctgate.MCTGate;

public class Circuit {

	// Private members
	private ArrayList<MCTGate> gate;
	private final int order;

	private boolean verifyOrder(int[] control) throws CircuitException {
		if (control == null || control.length != order)
			throw new CircuitException("Order of the gate and order of the circuit do not
match");
		return true;
	}

	public Circuit(int[] control) throws CircuitException {
		try {
			MCTGate firstGate = new MCTGate(control);
			gate = new ArrayList<>();
			gate.add(firstGate);
			order = firstGate.getOrder();
		} catch (GateException exp) {
			CircuitException cExp = new CircuitException("First gate cannot be created");
			cExp.initCause(exp);
```

```
                throw cExp;
            }
    }

    public MCTGate getGate(int index) throws CircuitException {
            if (index < 0 || index >= gate.size())
                    throw new CircuitException("Invalid gate number");
            return gate.get(index);
    }

    /**
     * Returns the number of input lines of the circuit.
     *
     * @return number of input lines
     */
    public int getOrder() {
            return order;
    }

    /**
     * Returns the number of gates in the circuit.
     *
     * @return number of gates present in the circuit.
     */
    public int noOfGates() {
            return gate.size();
    }

    /**
     * Loads the input of the circuit
     * Copies the content of the argument to the input buffer of the first gate of the circuit.
     * @param input boolean array containing inputs for the circuit input lines.
     * @throws CircuitException thrown if any internal {@link GateException} occurs
     *   while loading the input buffer of the {@link MCTGate}
     */

    public void loadInput(boolean[] input) throws CircuitException {
            try {
                    gate.get(0).setInputBuffer(input);
            } catch (GateException e) {
                    CircuitException cExp = new CircuitException("Input Cannot be loaded");
                    cExp.initCause(e);
                    throw cExp;
            }
    }
```

```
/**
 * Loads the output of the circuit.
 * Copies the content of the argument to the output buffer of the last gate of
 * the circuit. Useful for reverse propagation.
 * @param output boolean array containing inputs for the circuit output lines.
 * @throws CircuitException thrown if any internal {@link GateException} occurs
 *   while loading the output buffer of the {@link MCTGate}
 */
public void loadOutput(boolean[] output) throws CircuitException {
	try {
		gate.get(gate.size() - 1).setOutputBuffer(output);
	} catch (GateException e) {
		CircuitException cExp = new CircuitException("Output cannot be loaded");
		cExp.initCause(e);
		throw cExp;
	}
}

/**
 * Add a MCT gate.
 * This method adds a MCT gate to the circuit. The gate should be provided asinteger array.
 */
public void add(int[] control) throws CircuitException {
	// Check order
	verifyOrder(control);
	try {
		// Create gate
		MCTGate newGate = new MCTGate(control);
		// Add to the circuit
		gate.add(newGate);
	} catch (GateException e) {
		CircuitException cExp = new CircuitException("Gate cannot be added");
		cExp.initCause(e);
		throw cExp;
	}
}

/**
 * Clears all the buffers in the circuit.
 */
public void reset() {
	for (MCTGate g : gate) {
		g.clearBuffers();
	}
	}

	/**
	 * Propagate the input through the circuit.
	 */
	public void propagate() {
		int size = gate.size();
		try {
			for (int i = 0; i < size; i++) {
				if (i != 0)
					gate.get(i).setInputBuffer(gate.get(i - 1).getOutputBuffer());
				gate.get(i).propagate();
			}
		} catch (GateException exp) {
			reset();
			exp.printStackTrace();
		}
	}

	/**
	 * @see #propagate() propagate
	 */
	public void reversePropagate() {
		int size = gate.size();
		try {
			for (int i = size - 1; i >= 0; i--) {
				if (i != (size - 1))
					gate.get(i).setOutputBuffer(gate.get(i + 1).getInputBuffer());
				gate.get(i).reversePropagate();
				;
			}
		} catch (GateException exp) {
			reset();
			exp.printStackTrace();
		}
	}

	@Override
	public String toString() {
		StringBuilder str = new StringBuilder("");
		for (MCTGate mctGate : gate) {
			GateControl[] gC = mctGate.getControl();
			int[] intC =
Arrays.stream(gC).mapToInt(GateControl::gateControlToInt).toArray();
			str.append(Arrays.toString(intC) + "\n");
```

```
            }
            return str.toString();
        }
}
```

CircuitException.java :

Thrown for any possible exception occurring at Circuit class.

```
package reversiblecircuit.nctlibrary;
public class CircuitException extends Exception {
        private static final long serialVersionUID = 1L;
        public CircuitException(String message) {
                super(message);
        }
}
```

Test Set:

A $m \times n$ matrix, where
m= Number of inputs
n=number of test vector

Each test vector is a binary input to a reversible gate or a circuit.
Example:

$$\begin{array}{cc} & \text{TEST VECTOR} \\ \text{INPUT LINES} & \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Figure 28

Coding for test set:

Package for test set

TestSet.java :

Class to represent a test set.

```
package testset;

import java.util.ArrayList;
import java.util.Arrays;
import bit.*;

public class Testset {
        private ArrayList<boolean[]> testVector;
        private int order;
```

```
	/**
	 * Takes a boolean array and initialize a test set. Also set the order of the test set.
	 * @param vector boolean array representing the test vector.
	 * @throws TestsetException thrown if the argument is null or is an empty array
	 */
	public Testset(boolean[] vector) throws TestsetException {
		if (vector == null || vector.length == 0)
			throw new TestsetException("A testset obj must have atleast one test
vector");
		testVector = new ArrayList<>();
		testVector.add(vector);
		order = vector.length;
	}

	public boolean[] getTestVector(int index) throws TestsetException {
		if (index < 0 || index >= testVector.size())
			throw new TestsetException("Index out of bound");
		return testVector.get(index);
	}

	/**
	 * Returns the number of input lines of the test set
	 */
	public int getOrder() {
		return order;
	}

	/**
	 * Returns the number of test vectors in the test set.
	*/

	public int size() {
		return testVector.size();
	}

	/**
	 * Append a new test vector in the test set.
	 */
	public void add(boolean[] vector) throws TestsetException {
		if (vector == null || vector.length != order) {
			throw new TestsetException("Incompatible test vector");
		}
		testVector.add(vector);
	}
	@Override
	public String toString() {
		StringBuilder str = new StringBuilder("");
		int[] tVecInt;
		for (boolean[] tVec : testVector) {
			try {
				tVecInt = BitTransform.boolArrToIntArr(tVec);
				str.append(Arrays.toString(tVecInt) + "\n");
			} catch (BitTransformException e) {
				e.printStackTrace();
			}
		}
		return str.toString();
	}
}
package testset;
```

TestsetException.java :

Thrown for any possible exception occurring at Testset class.

```
	package testset;

	public class TestsetException extends Exception {
		private static final long serialVersionUID = 1L;

		/**
		 * Public constructor which takes a error message as a parameter.
		 */
		public TestsetException(String message) {
			super(message);
		}
	}
```

We work on missing gate fault detection and fault testing.

MISSING GATE FAULT DETECTION AND TESTING:

Testing of a reversible circuit, in general, turns out to be relatively simpler compared to that of non-reversible logic because of the inherent ease of controllability of logic states and observability of errors. Another important property that expedites the test generation process is the fact that back tracing is straightforward and always yields a unique vector at the input.

Single Missing Gate Fault (SMGF):

An SMGF is detected by setting logic 1 value on all the control inputs of the gate, and any value either 0 or 1 on the target input as well as on the wires not connected to the gate. In the example of this figure, if we apply {a, b, c} = {0, 1, 1} at the input of the circuit, the normal output would be {e, f, g} = {1, 0, 0}, whereas, in the presence of the SMGF fault marked by the dotted box, the output will be {e, f, g} = {0, 1, 1}. The number of possible SMGFs is equal to the number of gates in the circuit.

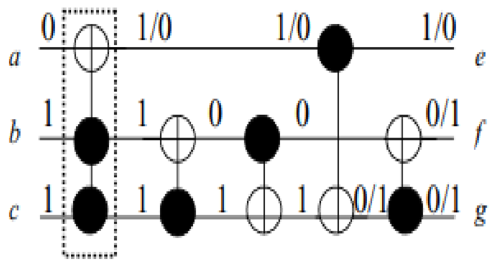


Figure 29

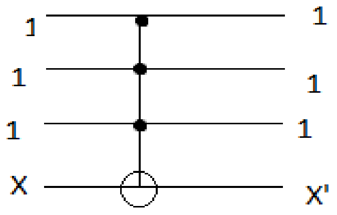


Figure 30 Fault free Circuit

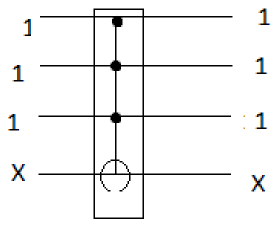


Figure 31 Faulty Circuit

A MCT gate is activated only when its all-control inputs are set to binary 1. Only this kind of input shows different behaviours (i.e, different output for the same input) in a fault free and a faulty circuit.

We can say a particular gate is tested only when it is feed with this kind of input.

ALGORITHM:

```
Procedure SMGF Testing (Circuit, Testset)

Let order = no. of rows in the Circuit.
Let no_of_gates = no of columns in the Circuit.
Let no_of_tests = no of columns in the Testset.
Let gate_result is an array of size no_of_gates which is initialized to false.
    If order =/= no. of rows in the Testset then
        Circuit and Testset are incompatible.
        Return
    [End if]
For t = 0 to no_of_tests - 1
Loop
     Load $t^{th}$ column of Testset to the Circuit input.
    Set ANDed_ctrl_ip = true.
    For g = 0 to no_of_gates - 1
    Loop
        Propagate through the $g^{th}$ gate.
      For i = 0 to order - 1
```

```
                Loop
                    If Circuit[i][g] = 1 then
                        If input at Circuit[i][g] is 1 then
                            ANDed_ctrl_ip = ANDed_ctrl_ip AND true
                        Else
                            ANDed_ctrl_ip = ANDed_ctrl_ip AND false
                        [End if]
                    [End if]
                [End loop]
                If ANDed_ctrl_ip = true then
                    gate_result[g] = true
                [End if]
        [End loop]
[End loop]
Set result = true
For r = 0 to no_of_gates
Loop
        result = result AND gate_result[r]
End loop
If result = true then
        Display Testset passed
Else
        Display Testset failed
[End if]
Return
```

Partial Missing Gate Fault (PMGF):

This is used to model the defects resulting from the partially misaligned or mistuned gate pulses.

A partial missing-gate fault (PMGF) corresponds to removing m of the k control inputs of a gate, thus transforming it into a (k – m)-MCT gate. The number m is called the order of the PMGF.

The detection criteria for a PMGF is that the missing control input is set to logic value 0 and we assign the logic value 1 to all other control inputs. The remaining input lines are assigned arbitrarily to logic value 0 and 1.

All k first-order PMGFs of a gate have detection conditions that conflict with each other and with the SMGF at the same gate, so k + 1 test vectors are required to test for all these faults. However, these vectors are guaranteed to also cover all the higher-order PMGFs at the same gate.

Example:

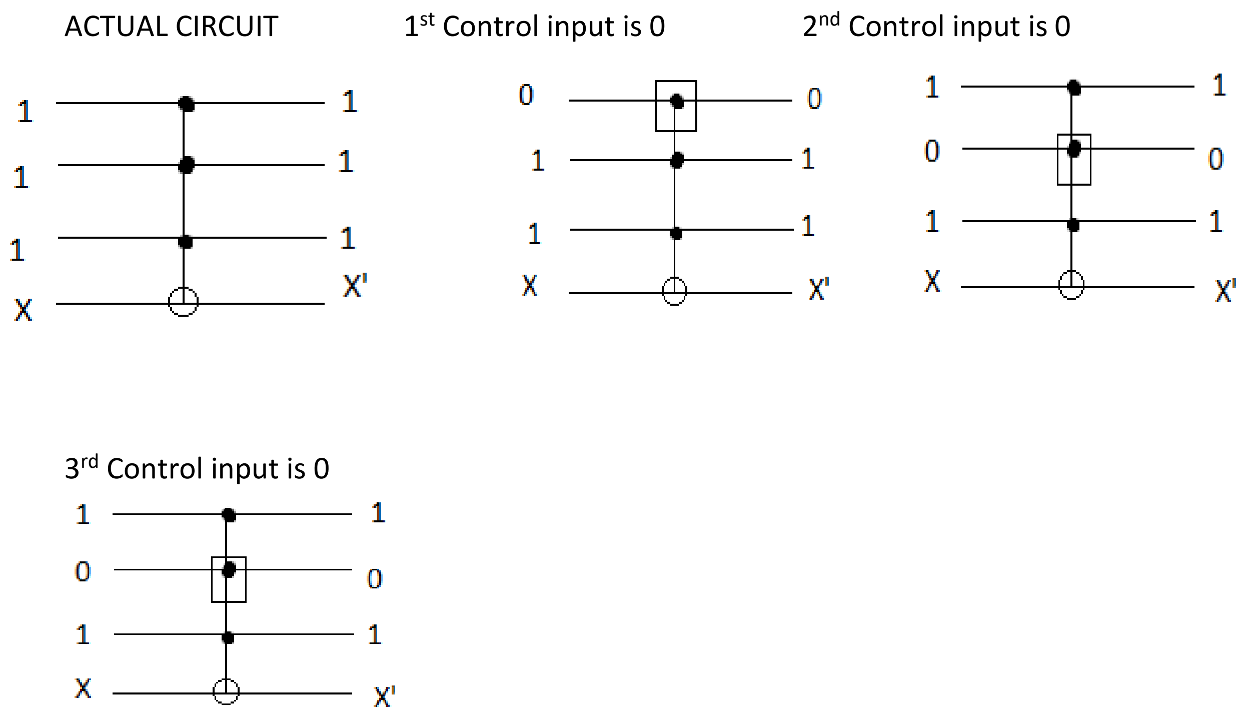


Figure 32 1st ORDER PMGF

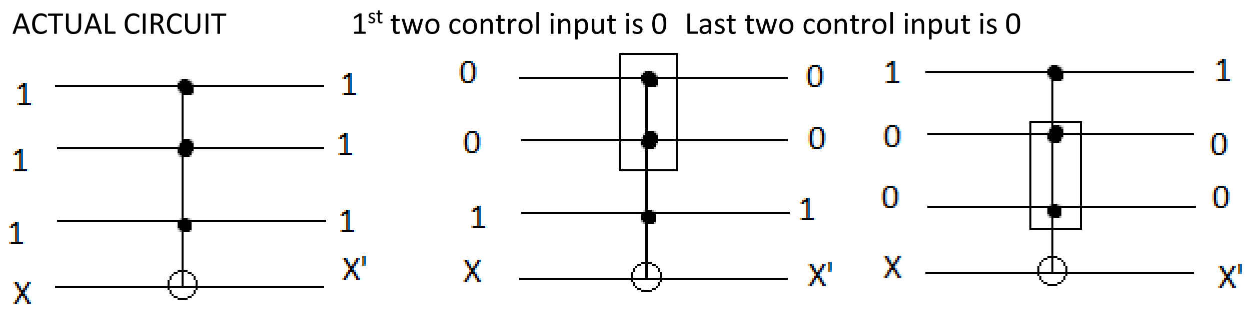


Figure 33 2nd ORDER PMGF

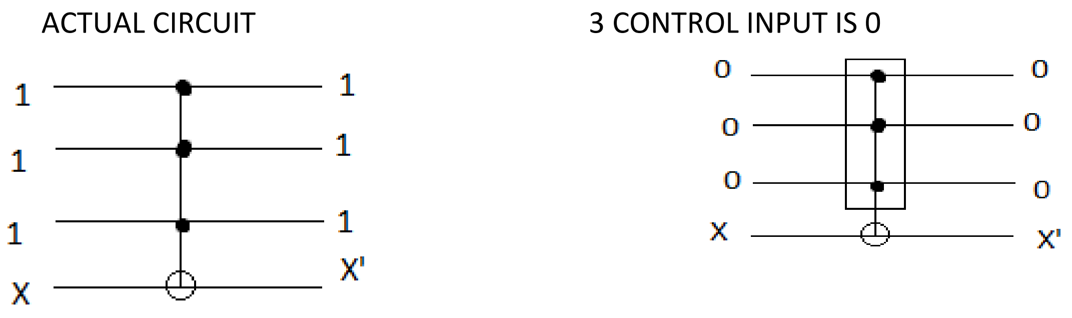


Figure 34 3rd ORDER PMGF

Data structure to be used for storing testing results-

Initially we will have two trees-one representing the circuit and the other one representing the test set.

1. For the representing a circuit-

   Root node represents the entire circuit

   Level 1 nodes represent the gates.

   Level 2 nodes represent control inputs of each gate.

Example:

CIRCUIT   Tree for the circuit

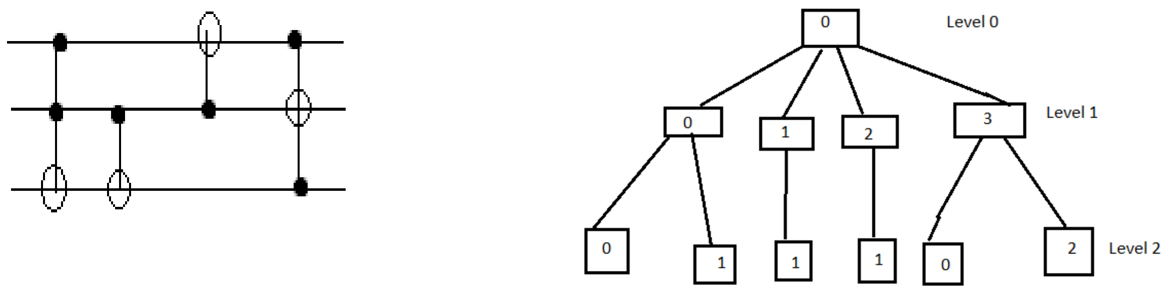


Figure 35 Circuit Diagram of PMGF

- Value of root node can be anything, we have chosen to make it 0.
- Value for level 1 children represents gate numbers starting from 0.
- Level 2 children values represent input lines where a gate has control inputs. For example. gate 0 has control input at line 0 and 1 (input line number also starts with 0) thus node 0 at level 1 has two children 0 and 1.

**** A node can have a value -1 to represent a CNOT gate which has no control input.

Eg.

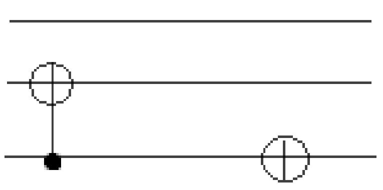


Figure 36 Demo Circuit

This circuit will have a tree.

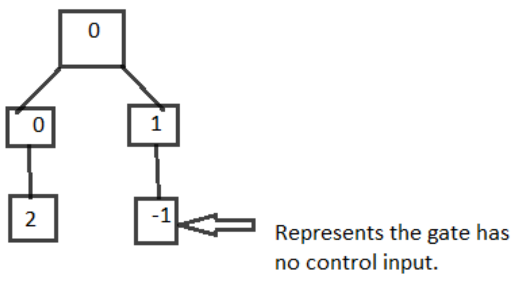


Figure 37 Tree for the above Circuit

2. For the tree representation of a test set—
    - Root node represents the entire test set.
    - Level 1 children represent test vectors of the test set.

Example:

Test set: Tree for the test set

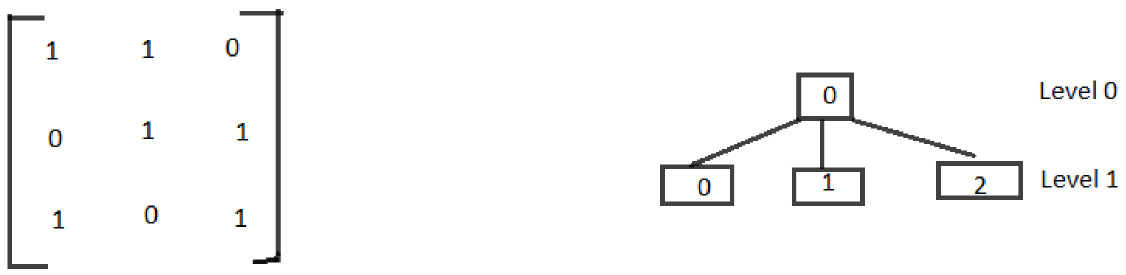


Figure 38 Test set Representation

- Value for root node can be anything, we have chosen to make it 0.
- Level 1 nodes are number -0,1,2 for test vector 0,1 and 2 respectively.

We would use these two trees to form a graph such that it shows which gate's control is tested by which test vector.

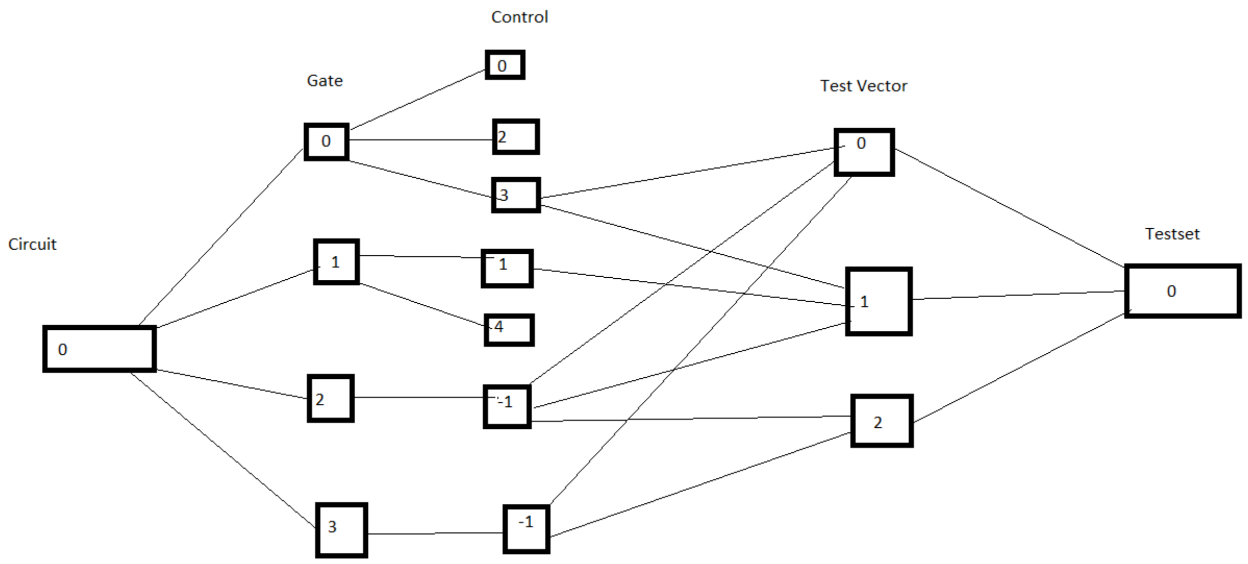


Figure 39 Data Structure of PMGF Fault Model Test

ALGORITHM:

Initialization of data structures for PMGF fault testing

Procedure PMGF_test (circuit_root, test set_root, circuit, test set)

```
Let no_of_gates = Number of gates in circuit
Let no_of_tests = Number of test vectors in a test set
Let order = Number of input lines in circuit

If order ≠ Number of input lines in test set then
        Display "Circuit and test set are incompatible"
        Return
[End if]

For t=0 to (no_of_tests – 1)
Loop
        Set test_ptr = test set_root.child(t) //Child(t) fetches the child node with value t.

        Get tth test vector from test set and store it in tVec.
        Load tVec to the input of circuit.
        Circuit.propagate()

        For g = 0 to (no_of_gates – 1)
        Loop
                Set candidate_ctrl = -1 // For the first control input of the gate which has     input
                binary 0

                Set rest_ctrl = true      // For computing the value of the control input.

                Set gate_ptr = circuit_root.child(9)

                Get 9th gate from circuit and store it in current_gate
                For p = 0 to (order – 1)
                Loop
                        If pth line of current gate is CONTROL INPUT then
                                If input 0 to pth line of current gate is binary 0 then
                                        If candidate_ctrl = -1 then
                                                candidate _ctrl = p
                                        Else
                                                Rest_ctrl = (rest_ctrl AND false)
                                        [End If]
                                Else
                                        rest_ctrl = (rest_ctrl AND true)
                                [End If]
                        [End If]
                [End Loop]
        If rest_ctrl = true then
            Set control_ptr = gate_ptr.child (candidate_ctrl)
            Link control_ptr and test_ptr
        [End If]
    [End Loop]
[End Loop]
Return
```

```
Procedure Show_result (circuit_root, circuit)
    Set no._of_gates = number of gates in circuit
    For g=0 to (no_of_gates – 1)
    Loop
        gate_ptr = circuit_root.child (9)
        For each child ctrl_ptr of gate_ptr
        Loop
            If ctrl_ptr has at least one child then
                For each child test_ptr of ctrl_ptr
                Loop
                    Display gate_ptr.value+ "gate's" + ctrl_ptr.value+ "control is
tested by vector:" + test_ptr.value
                [End Loop]
            Else
                result = result AND false
            [End If]
        [End Loop]
        If result = true then
            Display "Circuit is tested"
        Else
            Display "Circuit is not tested"
        [End If]
    [End Loop]
    Return
```

Coding for Missing Gate Fault (SMGF & PMGF) :

Package for MGF model :

MGF.java :

Class responsible for testing SMGF and PMGF fault provided a Circuit and Testset.

package faultmodel.mgf;

```
import java.util.Arrays;
import java.util.Map;

import faultmodel.mgf.pmgfdatastructure.PMGFException;
import faultmodel.mgf.pmgfdatastructure.PMGFResult;
import faultmodel.mgf.smgfdatastructure.SMGFException;
import faultmodel.mgf.smgfdatastructure.SMGFResult;
import reversiblecircuit.nctlibrary.Circuit;
import reversiblecircuit.nctlibrary.CircuitException;
import reversiblegate.mctgate.GateControl;
import testset.Testset;
import testset.TestsetException;

/**
 * This class is used to test Missing Gate Fault.
*/
public class MGF {
	private Circuit circuit;
	private Testset testset;
	private SMGFResult smgfResult;
	private PMGFResult pmgfResult;

	/**
	 * Logic for SMGF testing is implemented in this method
	 * @param tVecNo test vector number
	 * @param gateNo gate number
	 * @throws MGFException thrown for out of range arguments or any internal
	 */
	private void smgfLogic(int tVecNo, int gateNo) throws MGFException {
		if (tVecNo < 0 || tVecNo >= testset.size())
			throw new MGFException("Invalid test vector number");
		if (gateNo < 0 || gateNo >= circuit.noOfGates())
			throw new MGFException("Invalid gate number");
		try {
			GateControl[] controls = circuit.getGate(gateNo).getControl();
			boolean[] inBuff = circuit.getGate(gateNo).getInputBuffer();
			boolean andedControlInput = true;
			// Loop for each point in the gate
			for (int p = 0; p < circuit.getOrder(); p++)
				if (controls[p] == GateControl.CONTROL_INPUT)
					andedControlInput &= inBuff[p];
			smgfResult.setResult(tVecNo, gateNo, andedControlInput);
		} catch (CircuitException | SMGFException e) {
				MGFException exp = new MGFException("Internal circuit or SMGF
exception");
				exp.initCause(e);
				throw exp;
			}
		}

		/**
		 * Logic for PMGF testing is implemented in this method
		 *
		 * @param tVecNo test vector number
		 * @param gateNo gate number
		 * @throws MGFException thrown for out of range arguments or any internal
		 */
		private void pmgfLogic(int tVecNo, int gateNo) throws MGFException {
			if (tVecNo < 0 || tVecNo >= testset.size())
				throw new MGFException("Invalid test vector number");
			if (gateNo < 0 || gateNo >= circuit.noOfGates())
				throw new MGFException("Invalid gate number");

			try {
				GateControl curGateCtrl[] = circuit.getGate(gateNo).getControl();
				boolean ipBuff[] = circuit.getGate(gateNo).getInputBuffer();
				int noOfCtrl = 0;
				int candidateCtrl = -1;
				boolean restCtrl = true;
				int gateOrder = circuit.getOrder();
				for (int ipNo = 0; ipNo < gateOrder; ipNo++) {
					if (curGateCtrl[ipNo] == GateControl.CONTROL_INPUT) {
						noOfCtrl++;
						if (candidateCtrl == -1 && ipBuff[ipNo] == false)
							candidateCtrl = ipNo;
						else
							restCtrl &= ipBuff[ipNo];
					}
				}
				if (noOfCtrl == 0)
					pmgfResult.setForNoControl(gateNo, tVecNo);
				else if (candidateCtrl != -1 && restCtrl == true)
					pmgfResult.set(gateNo, candidateCtrl, tVecNo);
			} catch (CircuitException | PMGFException e) {
				MGFException exp = new MGFException("Internal circuit or PMGF
exception");
				exp.initCause(e);
				throw exp;
		}
	}

	/**
	 * It is required to provide a circuit and a test set to create a object of this class.
	 * After all validation an object is initialized. If the validation fails
	* @param tSet    test set to test with
	 * @throws MGFException thrown if the arguments are null or any internal
	 */
	public MGF(Circuit circuit, Testset tSet) throws MGFException {
		if (circuit == null || tSet == null)
			throw new MGFException("Either circuit is null or testset is null");
		if (circuit.getOrder() != tSet.getOrder())
			throw new MGFException("Circuit and testset are incompatible with each
other");
		this.circuit = circuit;
		this.testset = tSet;
		try {
			this.smgfResult = new SMGFResult(this.testset.size(),
this.circuit.noOfGates());
			this.pmgfResult = new PMGFResult(this.circuit, this.testset);
		} catch (SMGFException | PMGFException e) {
			MGFException exp = new MGFException("Internal SMGF or PMGF exception
occured");
			exp.initCause(e);
			throw exp;
		}
	}

	/**
	 * Tests the circuit using the test set.
	 */
	public void test() throws MGFException {
		int tCount = testset.size();
		int gCount = circuit.noOfGates();
		try {
			// Loop for each test vector
			for (int t = 0; t < tCount; t++) {
				circuit.loadInput(testset.getTestVector(t));
				circuit.propagate();
				// Loop for each gate
				for (int g = 0; g < gCount; g++) {
					smgfLogic(t, g);
					pmgfLogic(t, g);
				}
				}
			} catch (CircuitException | TestsetException e) {
				MGFException exp = new MGFException("Internal circuit or testset exception
occured");
				exp.initCause(e);
				throw exp;
			}
		}

		@Override
		public String toString() {
			StringBuilder str = new StringBuilder();

			int gCount = circuit.noOfGates();

			try {
				str.append("Gate result\n");
				for (int g = 0; g < gCount; g++) {
					str.append("Gate no: " + g + "\n");
					int[] smgfTVecs = smgfResult.gateToTVecs(g);
					str.append("SMGF Test Vectors: " + Arrays.toString(smgfTVecs) +
"\n");

					Map<Integer, int[]> pmgfCtrlTVecs =
pmgfResult.gateToGateCtrlTVecs(g);
					if (pmgfCtrlTVecs.get(pmgfResult.NO_CONTROL) == null) {
						str.append("PMGF Test Vectors: ");
						for (int key : pmgfCtrlTVecs.keySet()) {
							str.append("{" + key + ": " +
Arrays.toString(pmgfCtrlTVecs.get(key)) + "}");
						}
						str.append("\n");
					}
				}

				str.append("\nCircuit result\n");
				str.append("SMGF tested: " + smgfResult.isCircuitTested() + "\n");
				str.append("PMGF tested: " + pmgfResult.isCircuitTested() + "\n");

				return str.toString();
			} catch (SMGFException | PMGFException e) {
				e.printStackTrace();
			}

			return "";
```

```
	}
}
```

MGFException.java :

Thrown for any possible exception occurring at MGF class.

```
package faultmodel.mgf;

/**
 * Thrown for any possible exception occurring in the methods of the class
 */
public class MGFException extends Exception {
	private static final long serialVersionUID = 1L;

	public MGFException(String message) {
		super(message);
	}
}
```

Package for SMGF fault model :

Represents the data structures to store SMGF testing result.

SMGFResult.java :

Class stores the testing results to analyse latter.

```
package faultmodel.mgf.smgfdatastructure;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SMGFResult {
	private boolean[][] resultset;
	private final int tVCount, gateCount;

	/**
	 * Provided by number of tests and number of gates, it initializes the data
	 * structure to store the test result.
	 * Both the arguments should be valid(i.e. less than or equal to 0 not allowed)
	 * and must match with actual circuit and test set.
	*/
	public SMGFResult(int noOfTestVectors, int noOfGates) throws SMGFException {
		if (noOfTestVectors <= 0 || noOfGates <= 0)
			throw new SMGFException("Atleast one gate and one test vector is
requierd");
```

```
        resultset = new boolean[noOfTestVectors][noOfGates];
        tVCount = noOfTestVectors;
        gateCount = noOfGates;
}

/**
 * Updates the result.
 * If test vector t tests gate numbers g then call setResult(t, g, true)
 * otherwise call setResult(t, g, false) (although it is optional).
 */
public void setResult(int testVectorNo, int gateNo, boolean value) throws SMGFException {
        if (testVectorNo < 0 || testVectorNo >= tVCount)
                throw new SMGFException("Invalid test vector index");
        if (gateNo < 0 || gateNo >= gateCount)
                throw new SMGFException("Invalid gate index");
        resultset[testVectorNo][gateNo] = value;
}

/**
 * Clears the result.
 */
public void reset() {
        for (boolean[] tVec : resultset)
                Arrays.fill(tVec, false);
}

/**
 * Returns all the test vectors which test the gate, provided as argument.
 */
public int[] gateToTVecs(int gateNo) throws SMGFException {
        if (gateNo < 0 || gateNo >= gateCount)
                throw new SMGFException("Invalid gate index");
        List<Integer> result = new ArrayList<>();
        for (int t = 0; t < tVCount; t++) {
                if (resultset[t][gateNo])
                        result.add(t);
        }
        return result.stream().mapToInt(Integer::intValue).toArray();
}


public int[] tVecToGates(int tVecNo) throws SMGFException {
        if (tVecNo < 0 || tVecNo >= tVCount)
                throw new SMGFException("Invalid test vector index");
        List<Integer> result = new ArrayList<>();
```

```
		for (int g = 0; g < gateCount; g++) {
			if (resultset[tVecNo][g])
				result.add(g);
		}
		return result.stream().mapToInt(Integer::intValue).toArray();
	}

/**
 * Returns true if the entire circuit is tested.
* true if each and every gate of the circuit is tested by at least one test vector. Otherwise false.
*/
	public boolean isCircuitTested() {
		boolean[] gateResult = new boolean[gateCount];
		for (int t = 0; t < tVCount; t++) {
			for (int g = 0; g < gateCount; g++) {
				gateResult[g]  |= resultset[t][g];
			}
		}
		boolean result = true;
		for (int g = 0; g < gateCount; g++) {
			result &= gateResult[g];
		}
		return result;
	}
}
```

SMGFException.java :

Thrown for any possible exception occurring at SMGFResult class.

```
package faultmodel.mgf.smgfdatastructure;

/**
 * Thrown for any exception occurring at some of the methods of the class{@link SMGFResult}
 */
public class SMGFException extends Exception {
	private static final long serialVersionUID = 1L;

	/**
	 * Public constructor which takes a error message as a parameter.
	 * @param message of type {@link java.lang.String String}
	 */
	public SMGFException(String message) {
		super(message);
	}
}
```

Package for PMGF fault model :

Node.java :

Class used to define a node or vertex of a graph which will be further used in PMGFResult class.

```
package faultmodel.mgf.pmgfdatastructure;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Represents a node that is one individual element of a graph which is further  used by {@link
*PMGFResult} class to store PMGF test result.
 */
class Node {
	private Node parent;
	private int data;
	private Map<Integer, Node> child;

	/**
	 * Initialize a node with an integer data also initialize the parent node to
	 * null and child map to an empty map.
	 */
	public Node(int nodeData) {
		parent = null;
		data = nodeData;
		child = new HashMap<>();
	}

	/**
	 * Links the argument node with the invoking node.
	 */
	void setChildWithParent(int key, Node childNode) {
		if (childNode != null) {
			child.put(key, childNode);
			childNode.parent = this;
		}
	}

	/**
	 * Creates unidirectional link from invoking node to argument node
```

```
*/
void setOnlyChild(int key, Node childNode) {
        if (childNode != null) {
                child.put(key, childNode);
        }
}

/**
 * Returns the child node provided with the key of the child map.
*/
Node getChild(int key) {
        return child.get(key);
}

/**
 * Returns true if the node has at least one child.
 */
boolean haveChild() {
        return !child.isEmpty();
}

/**
 * Returns the number of children a node has
 */
int getNoOfChildren() {
        return child.size();
}

/**
 * Returns all the child key of the child map of the node.
 */
Set<Integer> getAllChildKey() {
        return child.keySet();
}

/**
 * Returns the data of the node.
 */
int getData() {
        return data;
}

/**
 * Returns the parent node of the invoking node.
 */
	Node getParent() {
		return parent;
	}
}
```

PMGFResult.java :

Class representing the entire data structure for storing PMGF testing result.

```
package faultmodel.mgf.pmgfdatastructure;

import java.util.HashMap;
import java.util.Map;

import reversiblecircuit.nctlibrary.Circuit;
import reversiblecircuit.nctlibrary.CircuitException;
import reversiblegate.mctgate.GateControl;
import testset.Testset;

/**
 * Data structure use to store the result of PMGF test.
*/
public class PMGFResult {
	private Node circuit, testset;
	public final int NO_CONTROL;

	/**
	 * Provided with a {@link Circuit} and a {@link Testset} it initialize the data structure.
	 */
	public PMGFResult(Circuit cir, Testset tset) throws PMGFException {
		if (cir == null || tset == null)
			throw new PMGFException("Either null circuit or null testset");
		NO_CONTROL = -1;

		try {
			circuit = new Node(0);
			// Node creation for each gate
			int noOfGates = cir.noOfGates();
			for (int g = 0; g < noOfGates; g++) {
				Node gateNode = new Node(g);
				// Node creation for each control
				GateControl[] control = cir.getGate(g).getControl();
				boolean haveControl = false;
				for (int c = 0; c < control.length; c++) {
					if (control[c] == GateControl.CONTROL_INPUT) {
						Node controlNode = new Node(c);
```

```
                                        gateNode.setChildWithParent(c, controlNode);
                                        haveControl = true;
                                }
                        }
                        if (haveControl == false) {
                                Node controlNode = new Node(NO_CONTROL);
                                gateNode.setChildWithParent(NO_CONTROL, controlNode);
                        }
                        // Gate node attachment to circuit node
                        circuit.setChildWithParent(g, gateNode);
                }
        } catch (CircuitException exp) {
                PMGFException pExp = new PMGFException("Circuit Exception occured");
                pExp.initCause(exp);
                throw pExp;
        }

        testset = new Node(0);
        int noOfTestVector = tset.size();
        for (int t = 0; t < noOfTestVector; t++) {
                Node tVecNode = new Node(t);
                testset.setChildWithParent(t, tVecNode);
        }
}

/**
 * Links a particular control input of a particular gate with a particular test vector.
 * Each control must be associated with a gate and thus three parameters are required.
 * @param gateNo    Gate number of the circuit.
 * @param controlNo Control point number of the gate.
 * @param tVecNo    Test vector number of the test set.
 * @throws PMGFException thrown if any of the arguments provided appears to be
 *                  invalid with respect to gate or test set.
 */
public void set(int gateNo, int controlNo, int tVecNo) throws PMGFException {
        Node gate = circuit.getChild(gateNo);
        if (gate == null)
                throw new PMGFException("Invalid gate number");
        Node control = gate.getChild(controlNo);
        if (control == null)
                throw new PMGFException("Invalid gate control");
        Node testVector = testset.getChild(tVecNo);
        if (testVector == null)
                throw new PMGFException("Invalid test vector number");
        control.setOnlyChild(tVecNo, testVector);
```

```
                testVector.setOnlyChild(gate.getData(), control);
        }

/**
 * It is similar to the {@link #set(int, int, int) set} method except it is used for CNOT gates.
* CNOT gate do not have any control input line, thus this method do not take any control number

*/
        public void setForNoControl(int gateNo, int tVecNo) throws PMGFException {
                set(gateNo, NO_CONTROL, tVecNo);
        }

/**
 * Provided with the gate number, it returns a {@link Map} containing information about which
*gate control line is tested by which test vectors.
* @throws PMGFException thrown for invalid gate number
 */

        public Map<Integer, int[]> gateToGateCtrlTVecs(int gateNo) throws PMGFException {
                Node gate = circuit.getChild(gateNo);
                if (gate == null)
                        throw new PMGFException("Invalid gate number");
                Map<Integer, int[]> result = new HashMap<>();
                for (int controlNo : gate.getAllChildKey()) {
                        result.put(controlNo,

        gate.getChild(controlNo).getAllChildKey().stream().mapToInt(Integer::intValue).toArray());
                }
                return result;
        }

/**
 * Provided with the test vector number, it returns a {@link Map} containing information about
*which gates and their controls are tested by the test vector.
 */
        public Map<Integer, Integer> tVecToGateGateCtrl(int tVecNo) throws PMGFException {
                Node tVec = testset.getChild(tVecNo);
                if (tVec == null)
                        throw new PMGFException("Invalid Test Vector number");
                Map<Integer, Integer> result = new HashMap<>();
                for (Integer gateNo : tVec.getAllChildKey()) {
                        result.put(gateNo, tVec.getChild(gateNo).getData());
                }
                return result;
        }
	/**
	 * Returns true if the entire circuit is tested otherwise returns false.
	 * <p>
	 * True if all the controls of all the gates are tested by at least one test vector.
	 */
	public boolean isCircuitTested() {
		for (Integer gateNo : circuit.getAllChildKey()) {
			Node gate = circuit.getChild(gateNo);
			for (Integer controlNo : gate.getAllChildKey())
				if (!gate.getChild(controlNo).haveChild())
					return false;
		}
		return true;
	}
}
```

PMGFException.java :

Thrown for any possible exception occurring at PMGFResult class.

```
package faultmodel.mgf.pmgfdatastructure;

/**
 * Thrown for any exception occurring at some of the methods of the classes
 * {@link Node} and {@link PMGFResult}
 */
public class PMGFException extends Exception {
	private static final long serialVersionUID = 1L;

	public PMGFException(String message) {
		super(message);
	}
}
```

Graphics User Interface

We have used java swing API to construct our application's user interface.

Package for GUI

Package component :

Holds the java swing components which is used latter used by the JFrame windows.

Button.java :

Represents individual button and there general properties.

```
package ui.component;
import java.awt.Font;
import javax.swing.JButton;

public class Button extends JButton {
	private static final long serialVersionUID = 1L;

	public Button(String title, int x, int y, int fontSize) {
		setText(title);
		setBounds(x, y, 200, 50);
		setFont(new Font("Cambria", Font.BOLD + Font.ITALIC, fontSize));
	}
}
```

Frame.java :

Represents the window which holds other components like button, labels etc.

```
package ui.component;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Frame extends JFrame {
	private static final long serialVersionUID = 1L;
	public Frame(String title) {
		setTitle(title);
		setLayout(null);
		setSize(1600, 900);
		setVisible(true);
		getContentPane().setBackground(new java.awt.Color(7, 74, 51));
		addWindowListener(new WindowAdapter() {

			@Override
			public void windowClosing(WindowEvent e) {
				int option = JOptionPane.showConfirmDialog(null, "Are you sure?",
"Close Information",
							JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
				if (option == JOptionPane.YES_OPTION)
					dispose();
```

```
                        }
                });
        }
}
```

Label.java :

Labels to show unchangeable data as well as output.

```
package ui.component;

import java.awt.Color;
import java.awt.Font;
import javax.swing.JLabel;

public class Label extends JLabel{
        private static final long serialVersionUID = 1L;

   public Label(String title, int width, int height, int fontSize) {
      setText(title);
      setSize(width, height);
      setFont(new Font("Cambria", Font.BOLD + Font.ITALIC, fontSize));
      setForeground(Color.WHITE);
   }
}
```

RadioButton.java :

Represents choice to select from different fault model.

```
package ui.component;

import java.awt.Color;
import java.awt.Font;

import javax.swing.JRadioButton;

public class RadioButton extends JRadioButton {
        private static final long serialVersionUID = 1L;

        public RadioButton(String title, int y) {
                setText(title);
                setBounds(100, y, 300, 40);
                setFont(new Font("Cambria", Font.BOLD + Font.ITALIC, 16));
                setBackground(new java.awt.Color(7, 74, 51));
                setForeground(Color.WHITE);
        }
}
```

Table.java :
Swing table to get circuit and testset data.

```
package ui.component;

import java.awt.Font;

import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.JTableHeader;

public class Table extends JTable {

	private static final long serialVersionUID = 1L;

	public Table(int row, int col) {
		super(row, col);
		SetTable(col);
	}

	public void SetTable(int col) {
		setFont(new Font("Cambria", Font.BOLD + Font.ITALIC, 16));
		setRowHeight(25);
		setCellSelectionEnabled(true);

		JTableHeader tableHeader = getTableHeader();
		tableHeader.setFont(new Font("Cambria", Font.BOLD + Font.ITALIC, 20));

		DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
		centerRenderer.setHorizontalAlignment(JLabel.CENTER);
		for (int i = 0; i < col; i++)
			getColumnModel().getColumn(i).setCellRenderer(centerRenderer);
	}

	public void loadTable(int[][] data) {
		int row = getRowCount();
		int col = getColumnCount();
		for(int r = 0; r < row; r++)
			for(int c = 0; c < col; c++) {
				setValueAt(Integer.valueOf(data[r][c]), r, c);
			}
	}
}
```

TextField.java :

Represents a text area for users to provide input to the application.

```
package ui.component;

import javax.swing.JTextField;

public class TextField extends JTextField {
	private static final long serialVersionUID = 1L;

	public TextField(int x, int y) {
		setBounds(x, y, 100, 20);
	}
}
```

Classes which hold each individual window of our application

HomeFrame.java :

Represents home screen and also holds reference to the data which will be used as well as transported by other frames like a circuit, a test set and a selected fault model.

```
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

import ui.component.Button;
import ui.component.Frame;
import ui.component.Label;

public class HomeFrame {

	private Frame mainFrame;
	private Label title;
	private Button circuit;
	private Button test_set;
	private Button fault_model;
	private Button check;
	private CircuitFrame cFrame;
	private TestsetFrame tFrame;
	private FaultModelFrame fMFrame;

	private int[][] circuitData;
	private int[][] testSetData;
	private String faultModel;

	public HomeFrame() {
		mainFrame = new Frame("Input GUI of Reversible Circuit");
		mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		title = new Label("Give the inputs for checking the faults in Reversible Circuit - ",
900, 150, 28);
		title.setHorizontalAlignment(JLabel.CENTER);
		mainFrame.add(title);

		circuit = new Button("Circuit Input", 350, 200, 16);
		mainFrame.add(circuit);

		test_set = new Button("Test Set Input", 350, 290, 16);
		mainFrame.add(test_set);

		fault_model = new Button("Fault Model Input", 350, 380, 16);
		mainFrame.add(fault_model);

		check = new Button("Check The Circuit", 650, 290, 16);
		mainFrame.add(check);

		circuitData = null;
		testSetData = null;
		faultModel = null;

		circuit.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				cFrame = new CircuitFrame();
			}
		});

		test_set.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				tFrame = new TestsetFrame();
				}
			});

			fault_model.addActionListener(new ActionListener() {

				@Override
				public void actionPerformed(ActionEvent e) {
					fMFrame = new FaultModelFrame();
				}
			});

			check.addActionListener(new ActionListener() {
				@Override
				public void actionPerformed(ActionEvent e) {
					if(cFrame == null || (circuitData = cFrame.getCircuitData()) == null)
						JOptionPane.showMessageDialog(mainFrame, "No circuit
provided");
					else if (tFrame == null || (testSetData = tFrame.getTestSetData()) ==
null)
						JOptionPane.showMessageDialog(mainFrame, "No test set
provided");
					else if ( fMFrame == null || (faultModel = fMFrame.getFaultModel())
== null || faultModel.isEmpty())
						JOptionPane.showMessageDialog(mainFrame, "No fault
model selected");
					else if (circuitData.length != testSetData.length)
						JOptionPane.showMessageDialog(mainFrame, "Incompatible
circuit and test set");
					else
						new OutputFrame(circuitData, testSetData, faultModel);
				}
			});
		}

		public void showCircuit() {
			if (circuitData == null)
				System.out.println("circuit is null");
			else {
				for (int r = 0; r < circuitData.length; r++) {
					for (int c = 0; c < circuitData[0].length; c++) {
						System.out.print(circuitData[r][c] + "\t");
					}
					System.out.println();
				}
			}
```

```
	}

	public void showTestSet() {
		if (testSetData == null)
			System.out.println("Test set is null");
		else {
			for (int r = 0; r < testSetData.length; r++) {
				for (int c = 0; c < testSetData[0].length; c++) {
					System.out.print(testSetData[r][c] + "\t");
				}
				System.out.println();
			}
		}
	}
}
```

CircuitFrame.java :

Represents the window where user will provide a circuit's data.

```
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;

import ui.component.Button;
import ui.component.Frame;
import ui.component.Label;
import ui.component.Table;
import ui.component.TextField;

public class CircuitFrame {

	private Frame circuitFrame;
	private Label circuitFrameHeading;
	private Label labelInputLine;
	private TextField inputField;
	private Label labelGateLine;
	private TextField gateField;
	private Button createButton;
	private Label tableTitle;
	private Label title0;
	private Label title1;
private Label title2;
private Table table;
private Button submit;
private Button back;
private JLabel gateTitle;

private int[][] circuitData;
private int row, column;

public CircuitFrame() {

        this.circuitData = null;
        this.row = 0;
        this.column = 0;

        circuitFrame = new Frame("Circuit Input of Reversible Circuit");

        circuitFrameHeading = new Label("Enter circuit data", 700, 100, 28);
        circuitFrameHeading.setHorizontalAlignment(JLabel.LEFT);
        circuitFrame.add(circuitFrameHeading);

        labelInputLine = new Label("Number of input lines", 200, 200, 16);
        circuitFrameHeading.setHorizontalAlignment(JLabel.LEFT);
        circuitFrame.add(labelInputLine);

        inputField = new TextField(120, 93);
        circuitFrame.add(inputField);

        labelGateLine = new Label("Number of gate", 200, 300, 16);
        circuitFrameHeading.setHorizontalAlignment(JLabel.LEFT);
        circuitFrame.add(labelGateLine);

        gateField = new TextField(120, 144);
        circuitFrame.add(gateField);

        createButton = new Button("create Table", 320, 100, 16);
        circuitFrame.add(createButton);

        tableTitle = new Label("Enter The Values Of The Table - ", 500, 500, 28);
        tableTitle.setHorizontalAlignment(JLabel.LEFT);
        tableTitle.setVisible(false);
        circuitFrame.add(tableTitle);

        gateTitle = new Label("Gate", 500, 500, 28);
        gateTitle.setHorizontalAlignment(JLabel.LEFT);
```

```
		gateTitle.setBounds(300, 50, 500, 500);
		gateTitle.setVisible(false);
		circuitFrame.add(gateTitle);

		title0 = new Label("0 -> Null Input", 200, 650, 20);
		title0.setHorizontalAlignment(JLabel.LEFT);
		title0.setBounds(10, 70, 200, 650);
		title0.setVisible(false);
		circuitFrame.add(title0);

		title1 = new Label("1 -> Control Input", 200, 750, 20);
		title1.setHorizontalAlignment(JLabel.LEFT);
		title1.setBounds(10, 100, 200, 650);
		title1.setVisible(false);
		circuitFrame.add(title1);

		title2 = new Label("2 -> Target Input", 200, 850, 20);
		title2.setHorizontalAlignment(JLabel.LEFT);
		title2.setBounds(10, 130, 200, 650);
		title2.setVisible(false);
		circuitFrame.add(title2);

		submit = new Button("Submit", 200, 550, 16);
		submit.setVisible(false);
		circuitFrame.add(submit);

		back = new Button("Back", 500, 550, 16);
		back.setVisible(true);
		circuitFrame.add(back);

		createButton.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {

				if (inputField.getText().isBlank())
					JOptionPane.showMessageDialog(circuitFrame, "Input line
field is empty");
				else if (Integer.parseInt(inputField.getText()) <= 0)
					JOptionPane.showMessageDialog(circuitFrame,
							"Number of input lines cannot be less than or
equal to 0");
				else if (gateField.getText().isBlank())
					JOptionPane.showMessageDialog(circuitFrame, "Gate field is
empty");
```

```
                                        else if (Integer.parseInt(gateField.getText()) <= 0)
                                                JOptionPane.showMessageDialog(circuitFrame, "Number of
gates cannot be less than or equal to 0");
                                        else {
                                                row = Integer.parseInt(inputField.getText());
                                                column = Integer.parseInt(gateField.getText());
                                                table = new Table(row, column);
                                                JScrollPane pane = new JScrollPane(table);
                                                pane.setBounds(200, 350, 300, 160);
                                                circuitFrame.getContentPane().add(pane);
                                                tableTitle.setVisible(true);
                                                table.setVisible(true);
                                                gateTitle.setVisible(true);
                                                title0.setVisible(true);
                                                title1.setVisible(true);
                                                title2.setVisible(true);
                                                submit.setVisible(true);
                                        }
                                }
                        });

                        submit.addActionListener(new ActionListener() {

                                @Override
                                public void actionPerformed(ActionEvent e) {
                                        boolean valid = true;
                                        OUTER_LOOP: for (int c = 0; c < column; c++) {
                                                int targetCount = 0;
                                                for (int r = 0; r < row; r++) {
                                                        if ((table.getValueAt(r, c) == null) || ((String)
table.getValueAt(r, c)).isBlank()) {

        JOptionPane.showMessageDialog(circuitFrame, "cell{" + r + "}{" + c + "} is empty");
                                                                valid = false;
                                                                break OUTER_LOOP;
                                                        } else if ((Integer.parseInt((String) table.getValueAt(r,
c)) < 0)
                                                                        || (Integer.parseInt((String)
table.getValueAt(r, c)) > 2)) {

        JOptionPane.showMessageDialog(circuitFrame,
                                                                                "cell{" + r + "}{" + c + "} contains
invalid value");
                                                                valid = false;
                                                                break OUTER_LOOP;
                                                } else if(Integer.parseInt((String) table.getValueAt(r,
c)) == 2) {
                                                        targetCount++;
                                                }
                                        }
                                        if(targetCount != 1) {
                                                JOptionPane.showMessageDialog(circuitFrame,
"Invalid number of target input at gate " + c);
                                                valid = false;
                                                break;
                                        }
                                }

                                if(valid == true) {
                                        circuitData = new int[row][column];
                                        for (int r = 0; r < row; r++) {
                                                for(int c = 0; c < column; c++) {
                                                        circuitData[r][c] = Integer.parseInt((String)
table.getValueAt(r, c));
                                                }
                                        }
                                        JOptionPane.showMessageDialog(circuitFrame, "Data saved
successfully");
                                        circuitFrame.dispose();
                                }
                        }
                });

                back.addActionListener(new ActionListener() {

                        @Override
                        public void actionPerformed(ActionEvent e) {
                                circuitFrame.dispose();
                        }
                });

        }

        public int[][] getCircuitData() {
                return circuitData;
        }

}
```

TestsetFrame.java :

Represents the window where user will provide a testset's data.

```
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;

import ui.component.Button;
import ui.component.Frame;
import ui.component.Label;
import ui.component.Table;
import ui.component.TextField;

public class TestsetFrame {

	private Frame testSetFrame;
	private Label tslabel;
	private Label inputLines;
	private Label testSet;
	private TextField inputField;
	private TextField testSetField;
	private Button create;
	private Label tstitle;
	private Table table;
	private Button submit;
	private Button back;

	private int[][] testSetData;
	private int row, column;
	private Label tVTitle;
	private Label inputTitle;

	public TestsetFrame() {
		testSetData = null;
		row = 0;
		column = 0;

		testSetFrame = new Frame("Test Set Input of Reversible Circuit");

		tslabel = new Label("Enter the Input Lines And Number of Test Set - ", 700, 100, 28);
		tslabel.setHorizontalAlignment(JLabel.LEFT);
```

```
testSetFrame.add(tslabel);

inputLines = new Label("Input Lines - ", 200, 200, 16);
tslabel.setHorizontalAlignment(JLabel.LEFT);
testSetFrame.add(inputLines);

testSet = new Label("Number of Test - ", 200, 300, 16);
tslabel.setHorizontalAlignment(JLabel.LEFT);
testSetFrame.add(testSet);

inputField = new TextField(160, 93);
testSetFrame.add(inputField);

testSetField = new TextField(160, 144);
testSetFrame.add(testSetField);

create = new Button("Create Table", 350, 100, 16);
testSetFrame.add(create);

tstitle = new Label("Enter The Values Of The Table - ", 500, 500, 28);
tstitle.setVisible(false);
testSetFrame.add(tstitle);

tVTitle = new Label("Test vectors", 500, 500, 28);
tVTitle.setHorizontalAlignment(JLabel.LEFT);
tVTitle.setBounds(400, 50, 500, 500);
tVTitle.setVisible(false);
testSetFrame.add(tVTitle);

inputTitle = new Label("Input lines(0 and 1 only)", 200, 650, 20);
inputTitle.setHorizontalAlignment(JLabel.LEFT);
inputTitle.setBounds(10, 70, 300, 650);
inputTitle.setVisible(false);
testSetFrame.add(inputTitle);

back = new Button("Back", 500, 600, 16);
back.setVisible(true);
testSetFrame.add(back);

submit = new Button("Submit", 200, 600, 16);
submit.setVisible(false);
testSetFrame.add(submit);

create.addActionListener(new ActionListener() {
					@Override
					public void actionPerformed(ActionEvent e) {
						if (inputField.getText().isBlank())
							JOptionPane.showMessageDialog(testSetFrame, "Input line
field is empty");
						else if (Integer.parseInt(inputField.getText()) <= 0)
							JOptionPane.showMessageDialog(testSetFrame,
									"Number of input lines cannot be less than or
equal to 0");
						else if (testSetField.getText().isBlank())
							JOptionPane.showMessageDialog(testSetFrame, "Test field is
empty");
						else if (Integer.parseInt(testSetField.getText()) <= 0)
							JOptionPane.showMessageDialog(testSetFrame, "Number of
tests cannot be less than or equal to 0");
						else {
							row = Integer.parseInt(inputField.getText());
							column = Integer.parseInt(testSetField.getText());
							table = new Table(row, column);
							JScrollPane pane = new JScrollPane(table);
							pane.setBounds(300, 350, 300, 160);
							testSetFrame.getContentPane().add(pane);
							tstitle.setVisible(true);
							tVTitle.setVisible(true);
							inputTitle.setVisible(true);
							table.setVisible(true);
							submit.setVisible(true);
						}
					}

				});

				submit.addActionListener(new ActionListener() {

					@Override
					public void actionPerformed(ActionEvent e) {
						boolean valid = true;
						OUTER_LOOP: for (int c = 0; c < column; c++) {
							for (int r = 0; r < row; r++) {
								if ((table.getValueAt(r, c) == null) || ((String)
table.getValueAt(r, c)).isBlank()) {

		JOptionPane.showMessageDialog(testSetFrame, "cell{" + r + "}{" + c + "} is empty");
									valid = false;
									break OUTER_LOOP;
```

```
                                        } else if ((Integer.parseInt((String) table.getValueAt(r,
c)) < 0)
                                                        || (Integer.parseInt((String)
table.getValueAt(r, c)) > 1)) {

        JOptionPane.showMessageDialog(testSetFrame,
                                                                "cell{" + r + "}{" + c + "} contains
invalid value");
                                                valid = false;
                                                break OUTER_LOOP;
                                        }
                                }
                        }

                        if (valid == true) {
                                testSetData = new int[row][column];
                                for (int r = 0; r < row; r++) {
                                        for (int c = 0; c < column; c++) {
                                                testSetData[r][c] = Integer.parseInt((String)
table.getValueAt(r, c));
                                        }
                                }
                                JOptionPane.showMessageDialog(testSetFrame, "Data saved
successfully");
                                testSetFrame.dispose();
                        }

                }
        });

        back.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                        testSetFrame.dispose();
                }
        });

    }

    public int[][] getTestSetData() {
        return testSetData;
    }

}
```

FaultModelFrame.java :

Represents the window for selecting a fault model.

```
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JOptionPane;

import ui.component.Button;
import ui.component.Frame;
import ui.component.Label;
import ui.component.RadioButton;

public class FaultModelFrame {

        private Frame faultModelFrame;
        private Label choose;
        private RadioButton MGFM;
        private RadioButton SAFM;
        private ButtonGroup singleSelection;
        private Button back;
        private Button submit;

        private String faultModel;

        public FaultModelFrame() {

                faultModel = null;

                faultModelFrame = new Frame("Fault Model Input of Reversible Circuit");

                choose = new Label("Choose the Fault Model from Below - ", 700, 100, 28);
                faultModelFrame.add(choose);

                MGFM = new RadioButton("Missing Gate Fault Model", 200);
                faultModelFrame.add(MGFM);

                SAFM = new RadioButton("Stuck-At Fault Model", 300);
                faultModelFrame.add(SAFM);

                singleSelection = new ButtonGroup();
```

```
		singleSelection.add(MGFM);
		singleSelection.add(SAFM);

		back = new Button("Back", 400, 450, 16);
		faultModelFrame.add(back);

		submit = new Button("Submit", 100, 450, 16);
		faultModelFrame.add(submit);

		submit.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				if (MGFM.isSelected())
					faultModel = "Missing Gate Fault Model";
				else if (SAFM.isSelected())
					faultModel = "Stuck-At Fault Model";
				else
					JOptionPane.showMessageDialog(null, "Select a fault 
Model");

				if (faultModel != null) {
					JOptionPane.showMessageDialog(null, "Saved Successfully");
					faultModelFrame.dispose();
				}
			}
		});

		back.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				faultModelFrame.dispose();
			}
		});
	}

	public String getFaultModel() {
		return faultModel;
	}
}
```

OutputFrame.java :

Window for showing the final test results.

```
package ui;
```

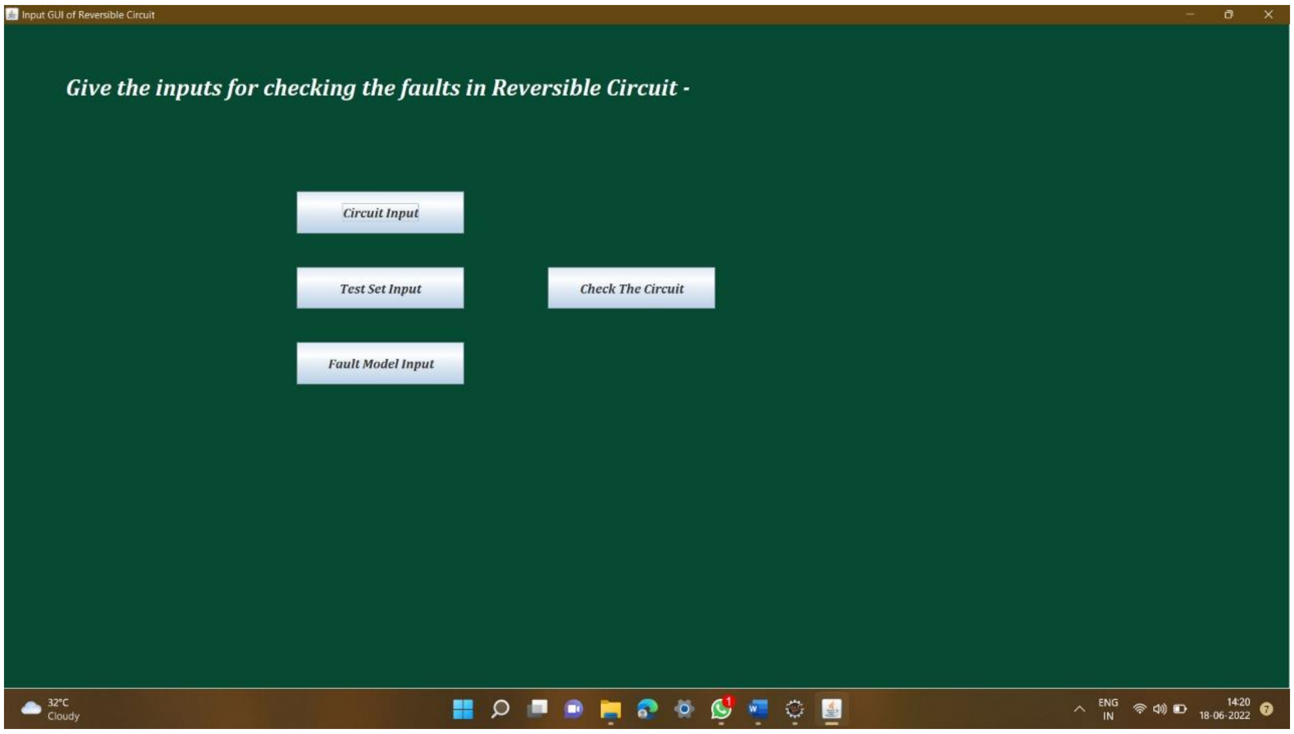
```
import javax.swing.JLabel;
import javax.swing.JScrollPane;

import bit.BitTransform;
import faultmodel.mgf.MGF;
import faultmodel.mgf.MGFException;
import reversiblecircuit.nctlibrary.Circuit;
import reversiblecircuit.nctlibrary.CircuitException;
import testset.Testset;
import testset.TestsetException;
import ui.component.Frame;
import ui.component.Label;
import ui.component.Table;

public class OutputFrame {

	private Frame checkFrame;
	private Label circuitLabel;
	private int circuitLabelcol;
	private Table cTable;
	private Label testSetLabel;
	private int testSetCol;
	private Table tsTable;
	private Label faultmodel;
	private Label fm;
	private Label output;

	private Circuit createCircuit(int[][] data) throws CircuitException {
		int row = data.length;
		int col = data[0].length;
		Circuit cir = null;
		for (int c = 0; c < col; c++) {
			int[] gate = new int[row];
			for (int r = 0; r < row; r++) {
				gate[r] = data[r][c];
			}
			if (c == 0)
				cir = new Circuit(gate);
			else
				cir.add(gate);
		}
		return cir;
	}

	private Testset createTestset(int[][] data) throws TestsetException {
		int row = data.length;
		int col = data[0].length;
		Testset tSet = null;
		for (int c = 0; c < col; c++) {
			boolean[] tVec = new boolean[row];
			for (int r = 0; r < row; r++) {
				tVec[r] = BitTransform.intToBool(data[r][c]);
			}
			if (c == 0)
				tSet = new Testset(tVec);
			else
				tSet.add(tVec);
		}
		return tSet;
	}

	public OutputFrame(int[][] circuit, int[][] testSet, String faultModel) {

		checkFrame = new Frame("Output GUI Of Reversible circuitLabel");

		circuitLabel = new Label("circuitLabel - ", 900, 100, 28);
		circuitLabel.setHorizontalAlignment(JLabel.LEFT);
		checkFrame.add(circuitLabel);

		circuitLabelcol = circuit[0].length;
		String[] circuitLabelcolumn = new String[circuitLabelcol];
		for (int i = 0; i < circuitLabelcol; i++)
			circuitLabelcolumn[i] = "Gate " + i;

		cTable = new Table(circuit.length, circuit[0].length);
		cTable.loadTable(circuit);
		JScrollPane circuitLabelpane = new JScrollPane(cTable);
		circuitLabelpane.setBounds(50, 80, 200, 100);
		checkFrame.getContentPane().add(circuitLabelpane);

		testSetLabel = new Label("Test Set - ", 900, 100, 28);
		testSetLabel.setHorizontalAlignment(JLabel.CENTER);
		checkFrame.add(testSetLabel);

		testSetCol = testSet[0].length;
		String[] testSetColumn = new String[testSetCol];
		for (int i = 0; i < testSetCol; i++)
			testSetColumn[i] = "Test Set" + i;

		tsTable = new Table(testSet.length, testSet[0].length);
		tsTable.loadTable(testSet);
		JScrollPane testSetLabelpane = new JScrollPane(tsTable);
		testSetLabelpane.setBounds(460, 80, 200, 100);
		checkFrame.getContentPane().add(testSetLabelpane);

		faultmodel = new Label("Fault Model - ", 900, 100, 28);
		faultmodel.setHorizontalAlignment(JLabel.RIGHT);
		checkFrame.add(faultmodel);

		fm = new Label(faultModel, 1100, 200, 24);
		fm.setHorizontalAlignment(JLabel.RIGHT);
		checkFrame.add(fm);

		try {
			String outputString;
			if(faultModel.equals("Stuck-At Fault Model"))
				outputString = "Under construction";
			else {
				Circuit newCircuit = createCircuit(circuit);
				Testset newTestset = createTestset(testSet);
				MGF mgfObj = new MGF(newCircuit, newTestset);
				mgfObj.test();
				outputString = "<html>" + mgfObj.toString().replaceAll("\n", "<br>");

			}
			output = new Label(outputString, 900, 500, 20);
			output.setHorizontalAlignment(JLabel.LEFT);
			output.setBounds(100, 200, 900, 500);
			checkFrame.add(output);
		} catch(CircuitException | TestsetException exp) {
			exp.printStackTrace();
		} catch (MGFException exp) {
			exp.printStackTrace();
		}
	}
}
```
Main.java :

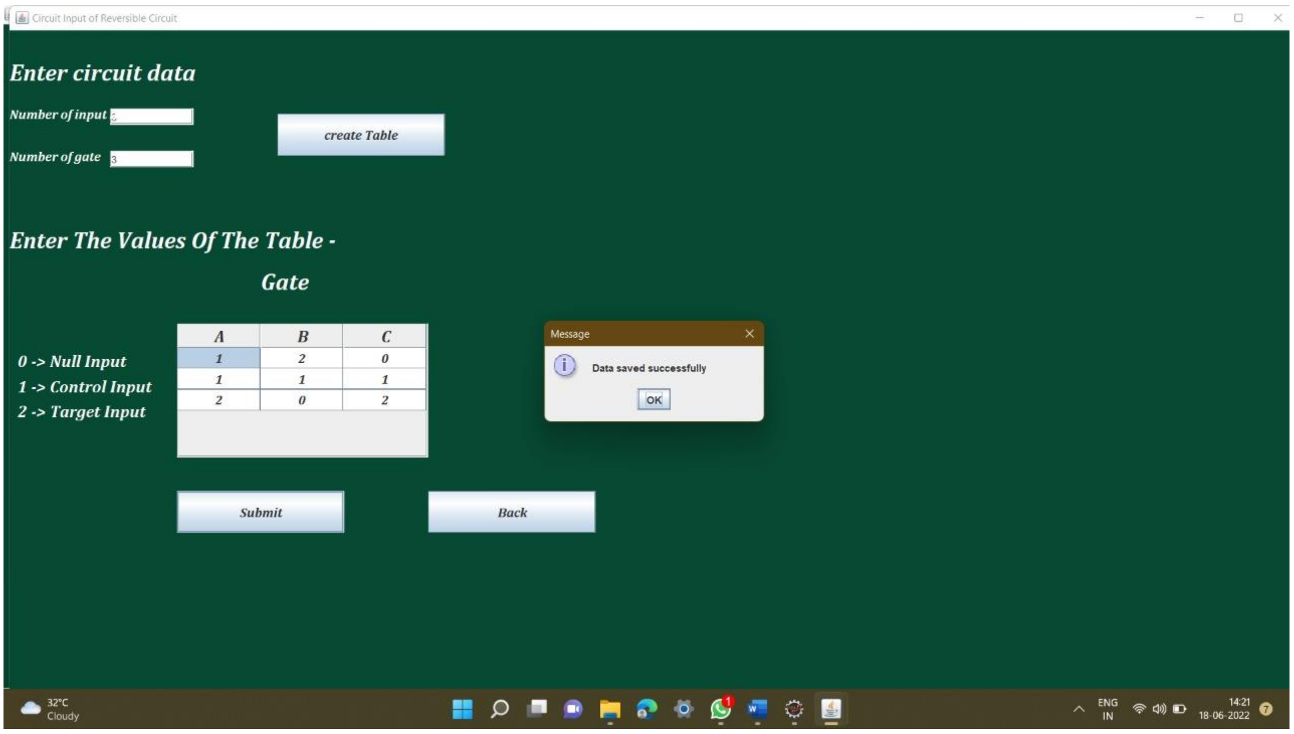
The entry point of the project.

```
package ui;
public class Main {
	public static void main(String[] args) {
		new HomeFrame();
	}
```

Screenshot of our application:

Overall execution:


Input GUI of Reversible Circuit
Give the inputs for checking the faults in Reversible Circuit -
Circuit Input
Test Set Input
Check The Circuit
Fault Model Input
32°C
Cloudy
ENG
IN
14:20
18-06-2022



Enter circuit data

Number of input 3

Number of gate 3

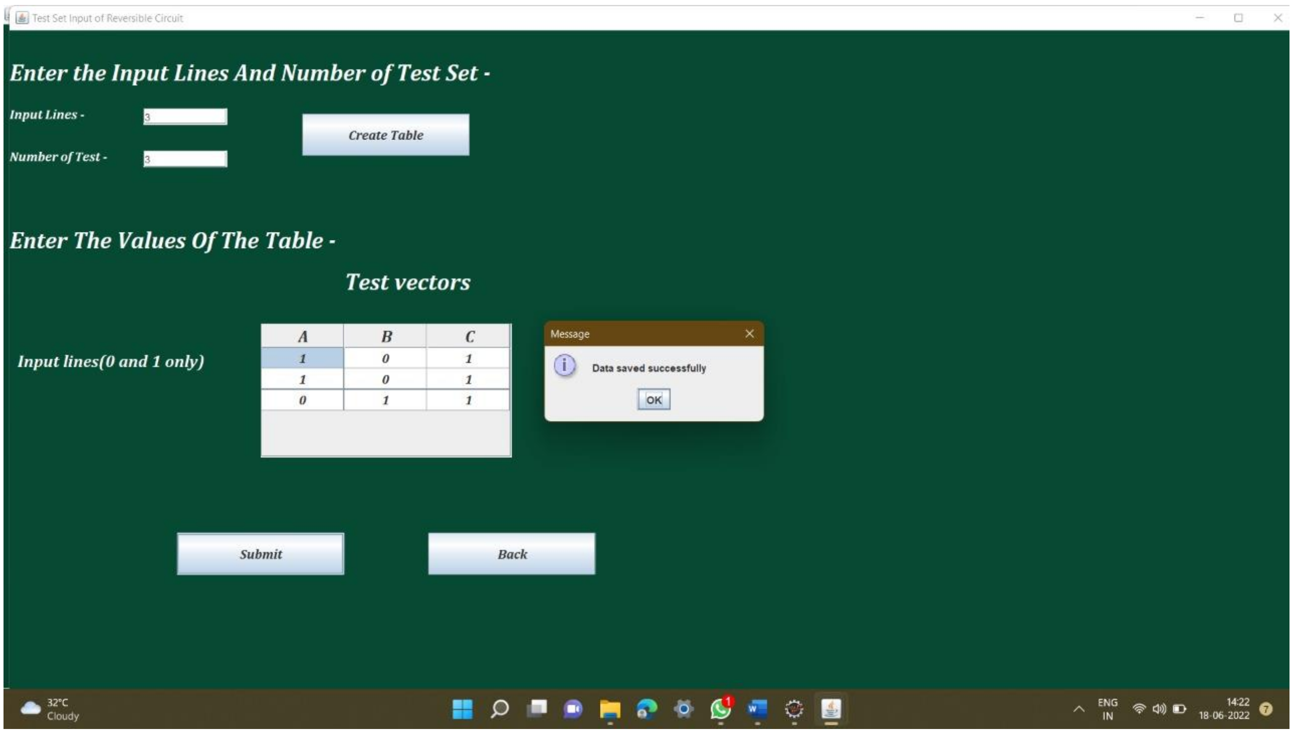
Enter The Values Of The Table -

Gate

0 -> Null Input
1 -> Control Input
2 -> Target Input

A	B	C
1	2	0
1	1	1
2	0	2

Data saved successfully



Test Set Input of Reversible Circuit

Enter the Input Lines And Number of Test Set -

Input Lines - 3

Number of Test - 3

Enter The Values Of The Table -

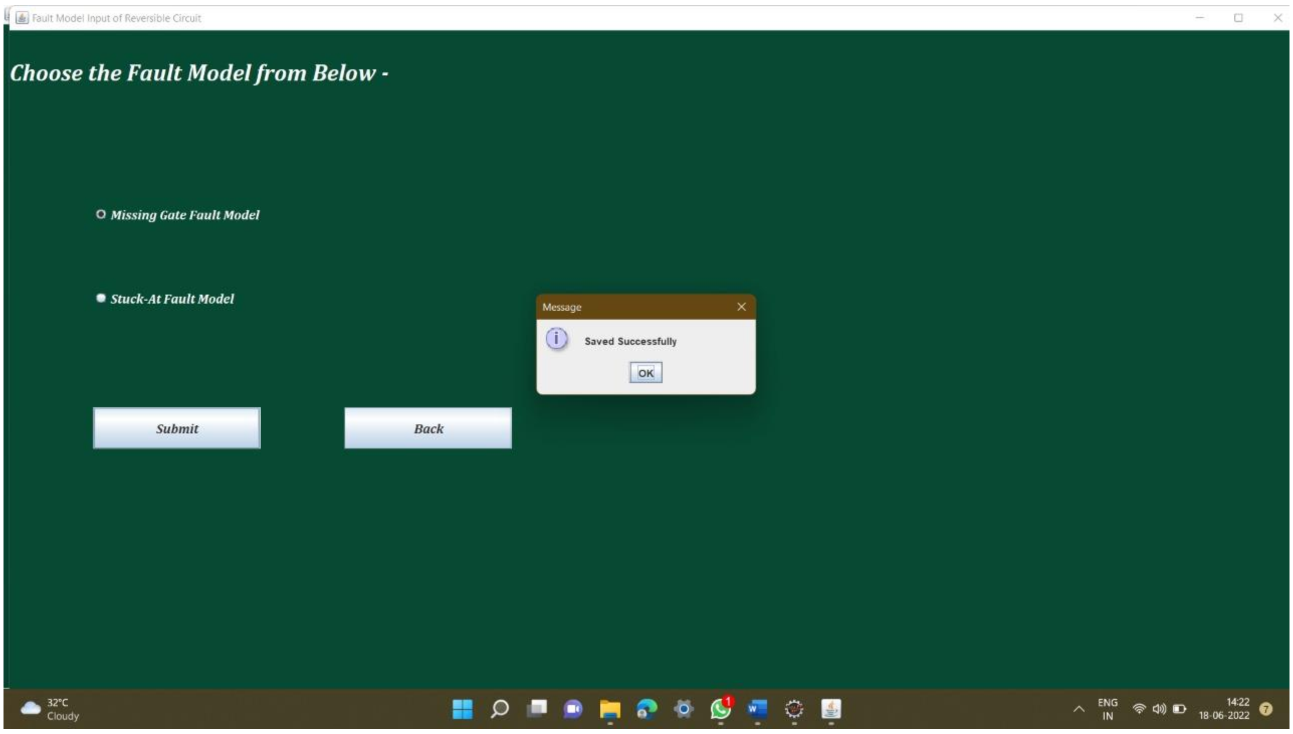
Test vectors

Input lines(0 and 1 only)

A	B	C
1	0	1
1	0	1
0	1	1

Message

Data saved successfully



Fault Model Input of Reversible Circuit

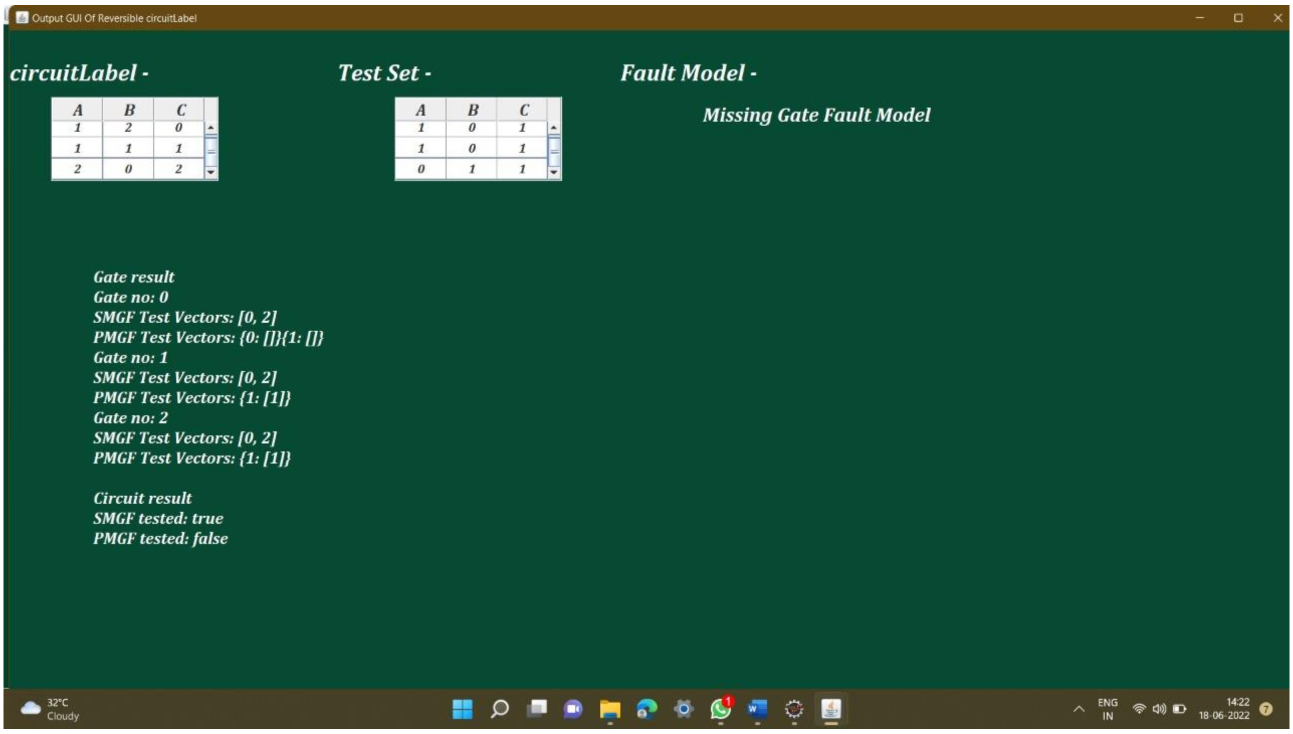
Choose the Fault Model from Below -

○ Missing Gate Fault Model

● Stuck-At Fault Model

Message

Saved Successfully


Output GUI Of Reversible circuitLabel

circuitLabel -

A	B	C
1	2	0
1	1	1
2	0	2

Test Set -

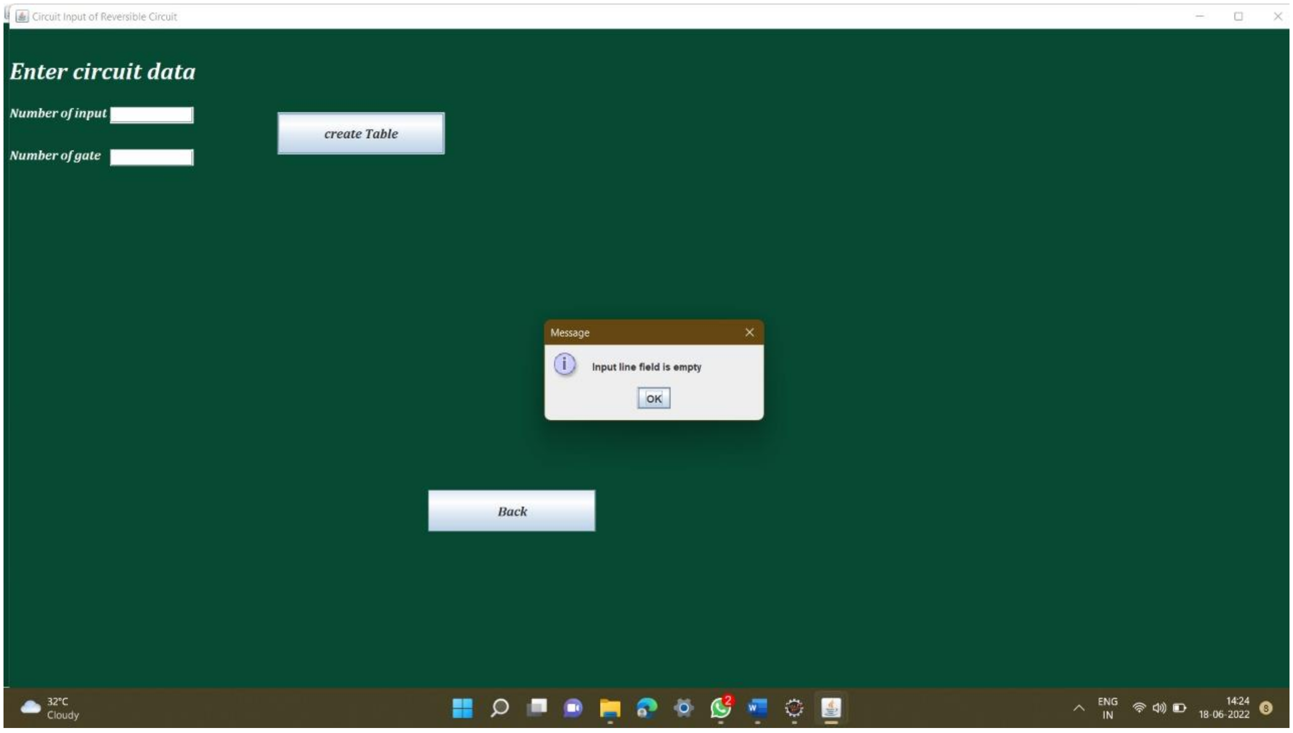
A	B	C
1	0	1
1	0	1
0	1	1

Fault Model -

Missing Gate Fault Model

```
Gate result
Gate no: 0
SMGF Test Vectors: [0, 2]
PMGF Test Vectors: {0: []}{1: []}
Gate no: 1
SMGF Test Vectors: [0, 2]
PMGF Test Vectors: {1: [1]}
Gate no: 2
SMGF Test Vectors: [0, 2]
PMGF Test Vectors: {1: [1]}

Circuit result
SMGF tested: true
PMGF tested: false
```

Validation:


Circuit Input of Reversible Circuit

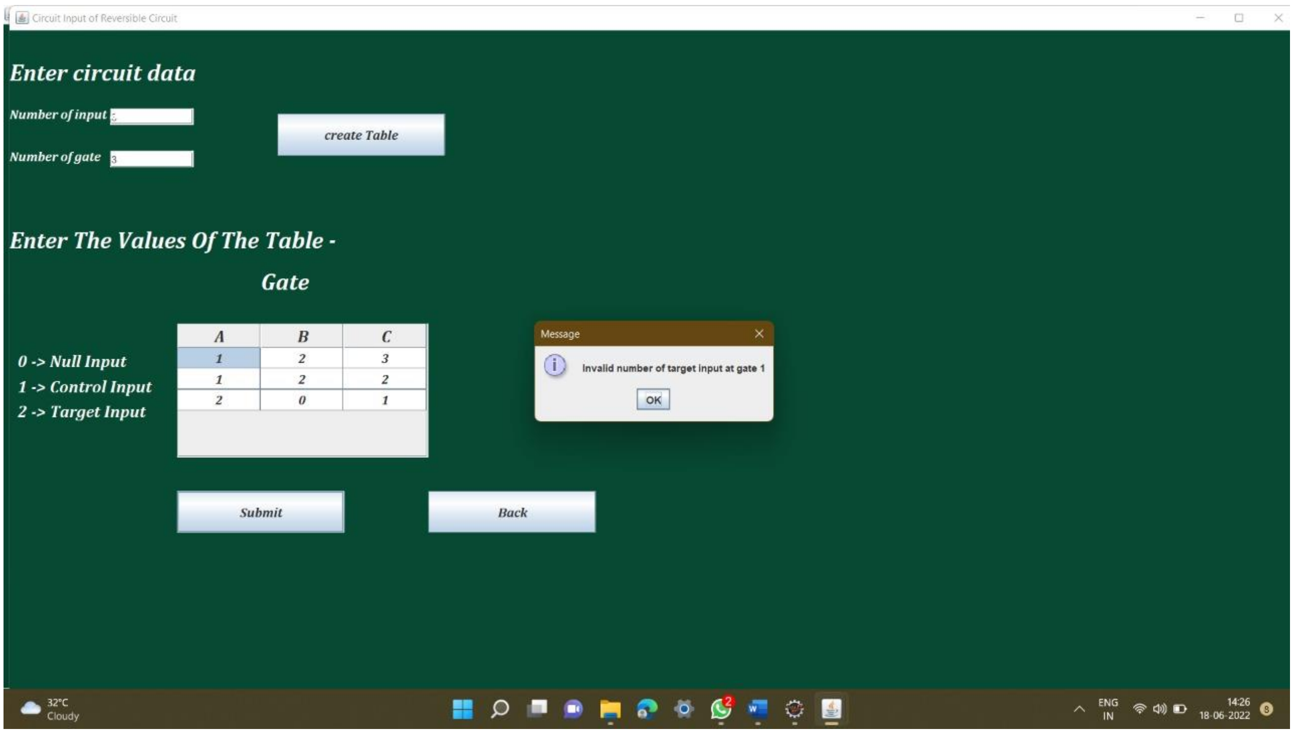
Enter circuit data

Number of input

Number of gate

Message

Input line field is empty


Circuit Input of Reversible Circuit

Enter circuit data

Number of input 3

Number of gate 3

Enter The Values Of The Table -

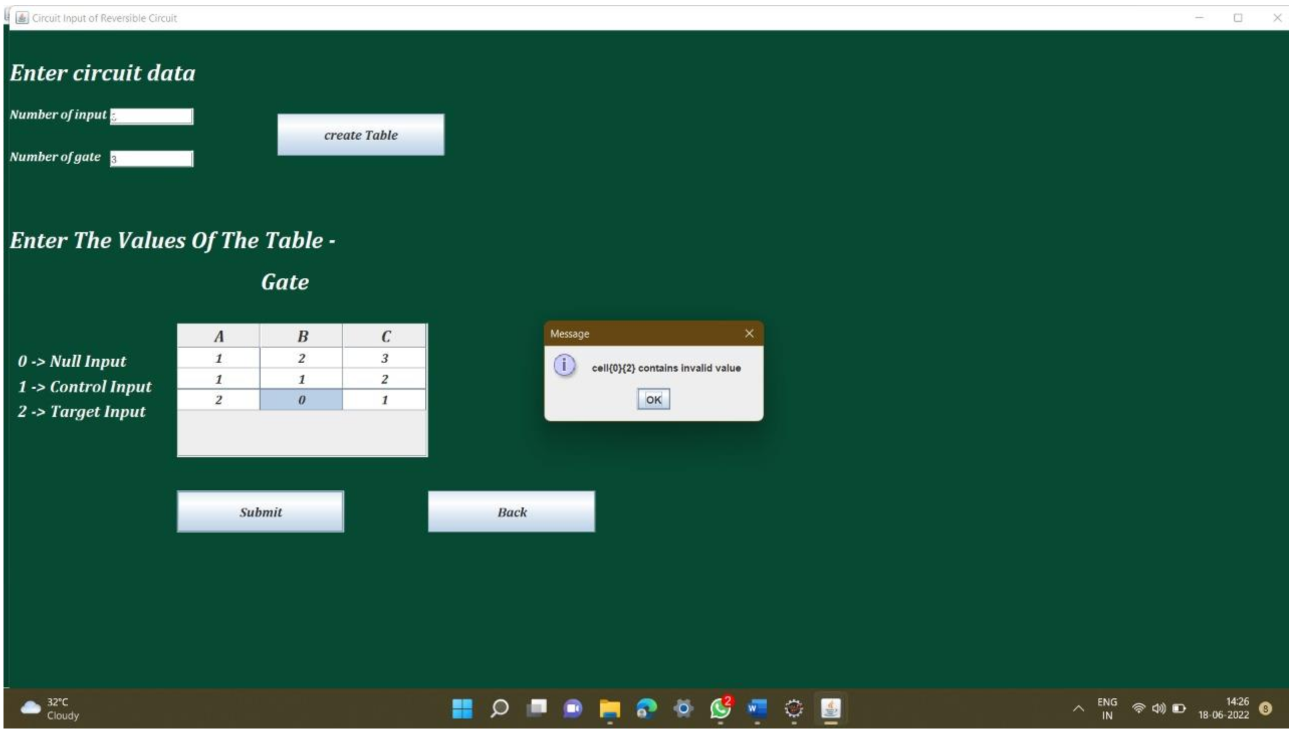
Gate

0 -> Null Input
1 -> Control Input
2 -> Target Input

A	B	C
1	2	3
1	2	2
2	0	1

Message

Invalid number of target input at gate 1


Circuit Input of Reversible Circuit

Enter circuit data

Number of input 3

Number of gate 3

Enter The Values Of The Table -

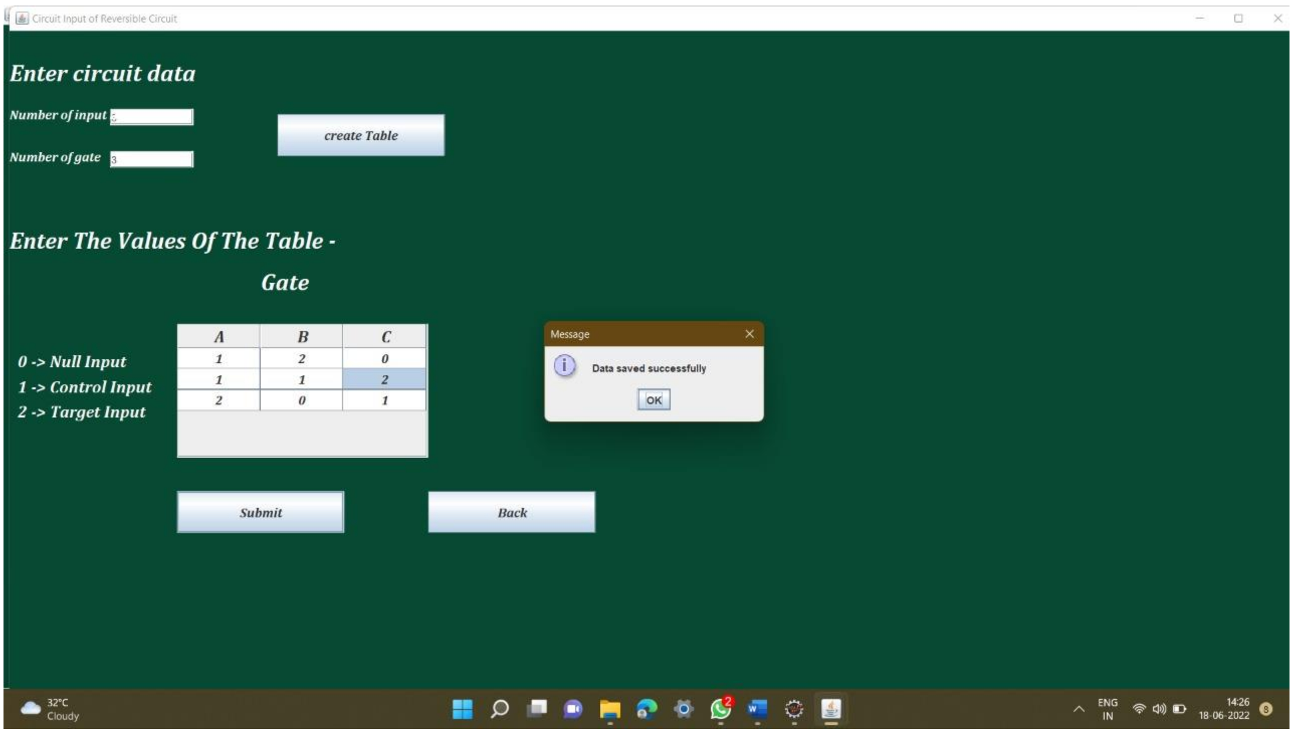
Gate

0 -> Null Input
1 -> Control Input
2 -> Target Input

A	B	C
1	2	3
1	1	2
2	0	1

Message

cell{0}{2} contains invalid value



Circuit Input of Reversible Circuit

Enter circuit data

Number of input 3

Number of gate 3

create Table

Enter The Values Of The Table -

Gate

0 -> Null Input
1 -> Control Input
2 -> Target Input

A	B	C
1	2	0
1	1	2
2	0	1

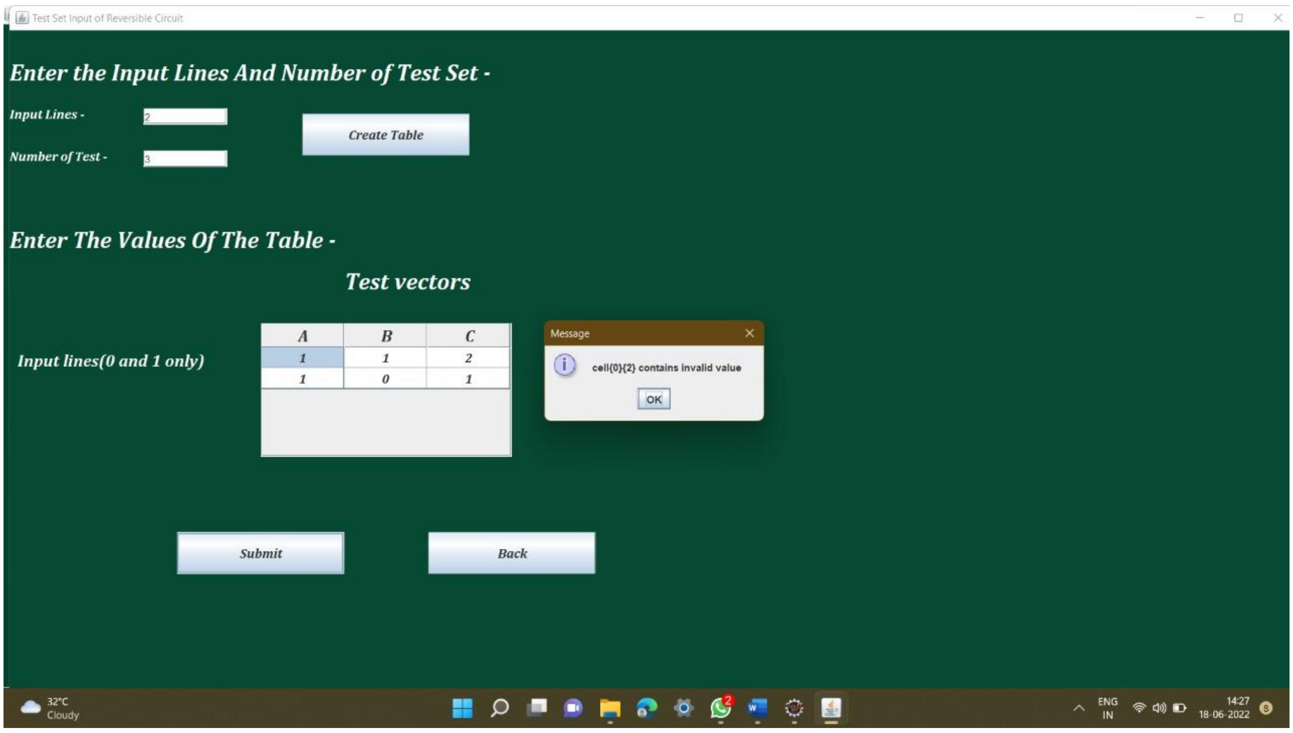
Message

Data saved successfully

OK

Submit

Back



Test Set Input of Reversible Circuit

Enter the Input Lines And Number of Test Set -

Input Lines - 2

Number of Test - 3

Create Table

Enter The Values Of The Table -

Test vectors

Input lines(0 and 1 only)

A	B	C
1	1	2
1	0	1

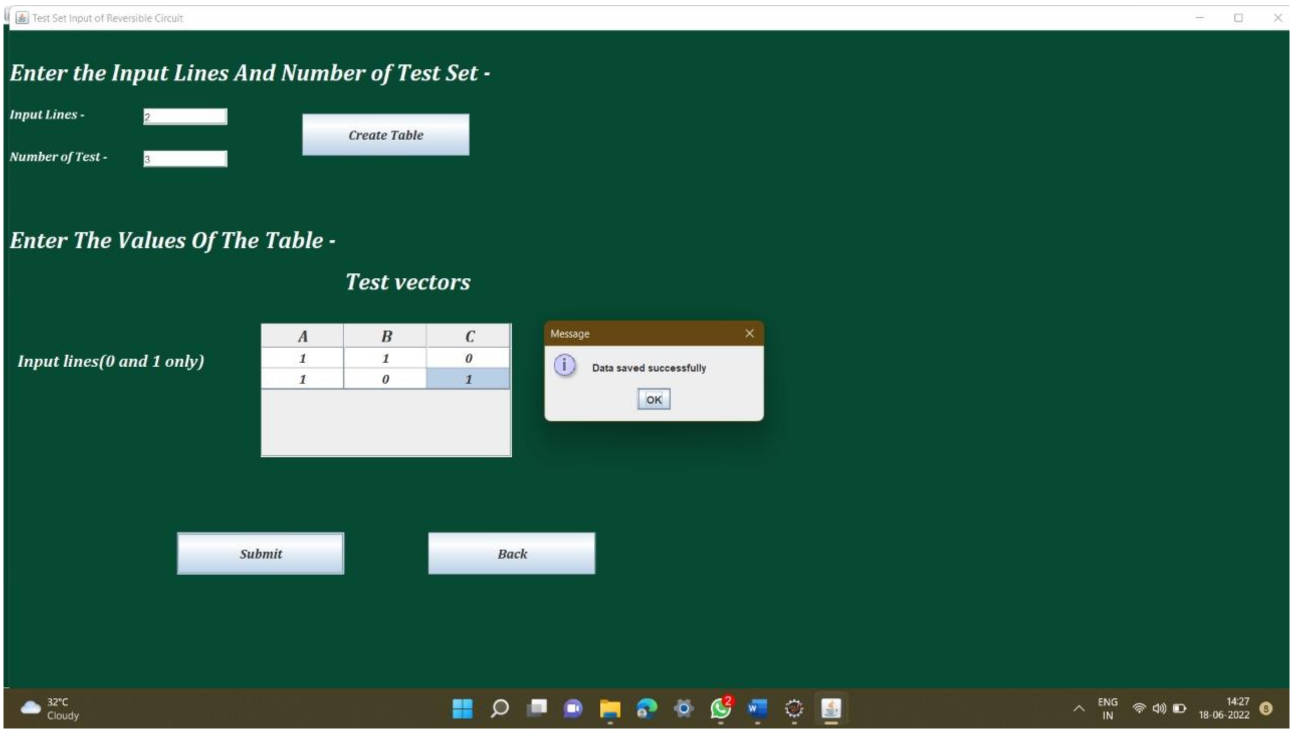
Message

cell{0}{2} contains invalid value

OK

Submit

Back


Test Set Input of Reversible Circuit

Enter the Input Lines And Number of Test Set -

Input Lines - 2

Number of Test - 3

Create Table

Enter The Values Of The Table -

Test vectors

Input lines(0 and 1 only)

A	B	C
1	1	0
1	0	1

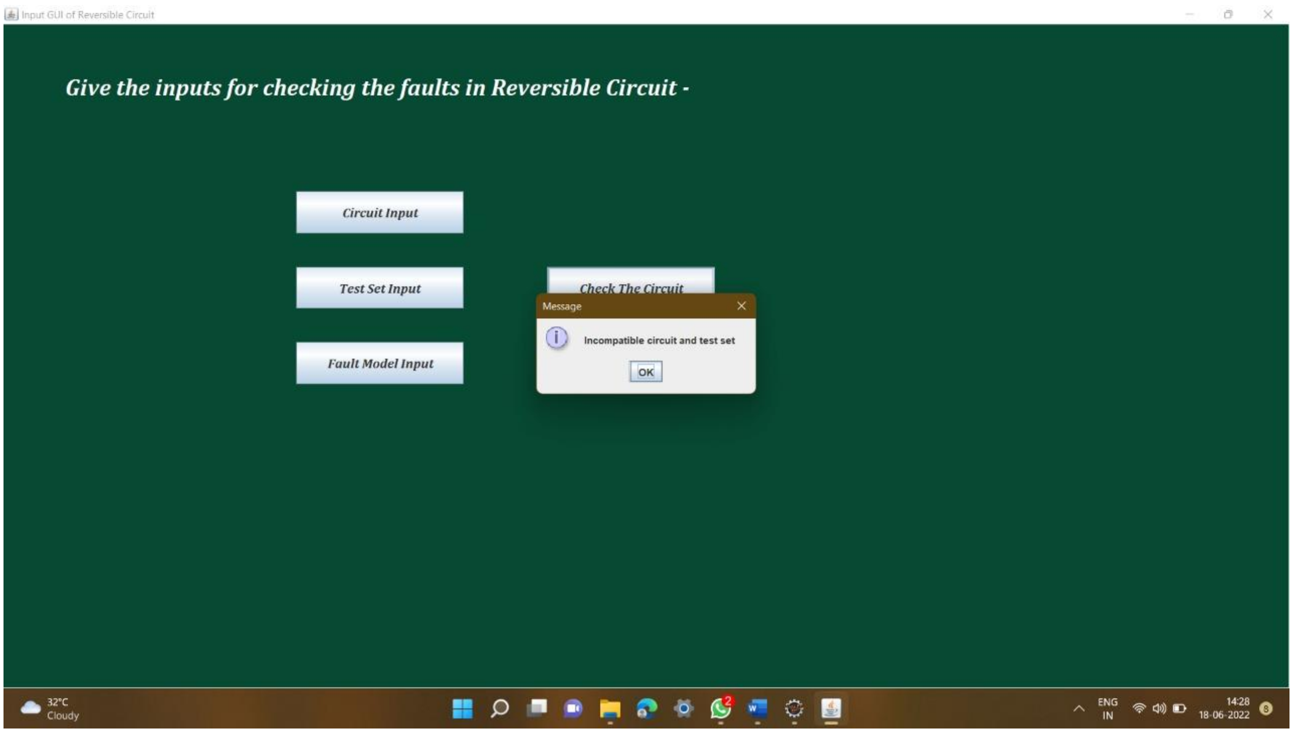
Message

Data saved successfully

OK

Submit

Back


Input GUI of Reversible Circuit

Give the inputs for checking the faults in Reversible Circuit -

Circuit Input

Test Set Input

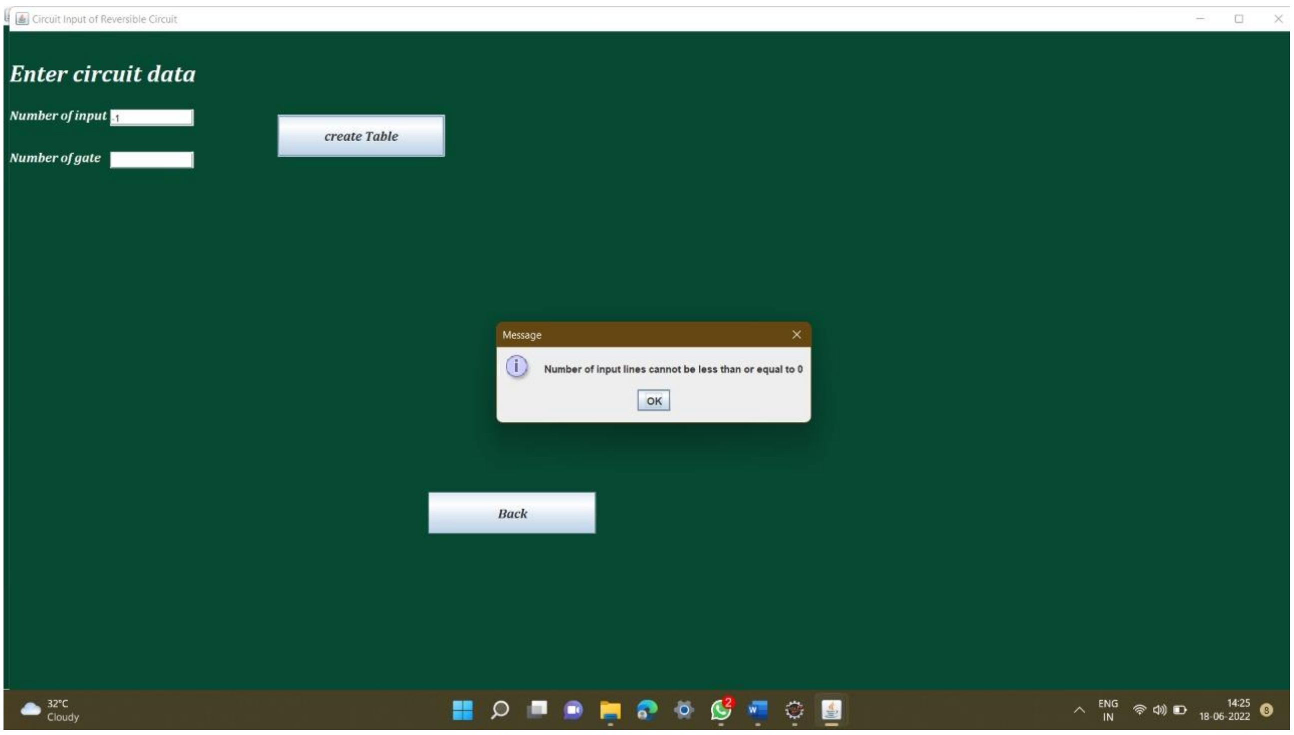
Fault Model Input

Check The Circuit

Message

Incompatible circuit and test set

OK



Number of input lines cannot be less than or equal to 0

SUMMARY:

From the theoretical concept of reversible circuit we have created a tool which can take a circuit and a test set and then checks whether the test set can test all the possible SMGF and PMGF faults of the circuit. In order to create a circuit we have to instantiate the Circuit class and then add MCT gates to it. In order to create a test set we have to instantiate the Testset class and then add test vectors to it. After the creation of circuit and test set we can create an object of MGF class and pass this circuit and test set to it. The MGF object has a test method which will give us the desired output for both SMGF and PMGF faults. This output is a string which describes all the detected as well as undetected fault in the circuit.

FUTURE WORK:

So far, our tool can detect SMGF and PMGF fault. Our aim will be to add MMGF (Multiple Missing Gate Fault) and RGF (Repeated Gate Fault) in our tool. We are also trying to add other different fault models for example Stuck-at Fault, Bridging Fault etc. Not just that we are also trying to make our user interface more attractive and interactable.

REFERENCES:

1.R. Landauer, “Irreversibility and heat generation in the computing process,” IBM Research and Development, vol. 5, pp. 183-191, 1961.
2. M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information. New York: Cambridge Univ. Press, 2000.
3. K. N. Patel, J. P. Hayes, and I. L. Markov, “Fault testing for reversible circuits,” VTS, pp. 410-416, 2003.
4.J.-S. Chang and C.-S. Lin, “Test set compaction for combinational circuits,” IEEE Trans. Computer-Aided Design, vol. 14, pp. 1370–1378, Nov. 1995
5. J.-S. Chang and C.-S. Lin, “Test set compaction for combinational circuits,” IEEE Trans. Computer-Aided Design, vol. 14, pp. 1370–1378, Nov. 1995
6.D. P. Vasudevan, P. K. Lala and J. P. Parkerson, “A novel approach for on-line testable reversible logic circuit design,” ATS 2004, pp. 325-330.
7. A. Chakraborty, “Synthesis of reversible circuits for testing with universal test set and C-testability of reversible iterative logic arrays,” VLSI Design, pp. 249-254, 2005.
8. H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya, “Detection of bridging faults in a reversible circuit”, VLSI Design and Test, pp. 384-392, 2006.
9. H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya, “Minimal test set for bridging fault detection in reversible circuits”, ATS, Bejing, China, pp. 125-128, 2007.
10. I. Polian, J. P. Hayes, T. Fienn, and B. Becker, “A family of logical fault models for reversible circuits”, ATS, Kolkata, India, 2005, pp. 422-427.
11.R. Wille, S. Offermann, and R. Drechsler, “SyReC: A programming language for synthesis of reversible circuits,” in Forum on Specification and Design Languages, 2010, pp. 184–189.
12. M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler,“Synthesis of reversible circuits with minimal lines for large functions,” in ASP Design Automation Conf., 2012, pp. 85–92.
13. D. Y. Feinstein, M. A. Thornton, and D. M. Miller, “Partially redundant logic detection using symbolic equivalence checking in reversible and irreversible logic circuits,” in Design, Automation and Test in Europe, 2008, pp. 1378–1381.
14. D. M. Miller, R. Wille, and R. Drechsler, “Reducing reversible circuit cost by adding lines,” in Int’l Symp. on Multi-Valued Logic, 2010, pp. 217–222.