IMPLEMENTATION OF COMPACTION ALGORITHM FOR ATPG GENERATED PARTIALLY SPECIFIED TEST DATA

Vaishali Dhare[1] and Dr. Usha Mehta[2]

[1]Assistant Professor, Institute of Technology, Nirma University, Ahmedabad
[2]Senior Associate Professor, Institute of Technology,
Nirma University, Ahmedabad
[1]vaishalidhare@nirmauni.ac.in [2]usha.mehta@nirmauni.ac.in

ABSTRACT :

In this paper the ATPG is implemented using C++. This ATPG is based on fault equivalence concept in which the number of faults gets reduced before compaction method. This ATPG uses the line justification and error propagation to find the test vectors for reduced fault set with the aid of controllability and observability. Single stuck at fault model is considered. The programs are developed for fault equivalence method, controllability Observability, automatic test pattern generation and test data compaction using object oriented language C++. ISCAS 85 C17 circuit was used for analysis purpose along with other circuits. Standard ISCAS (International Symposium on Circuits And Systems) netlist format was used. The flow charts and results for ISCAS 85 C17 circuits along with other netlists are given in this paper. The test vectors generated by the ATPG further compacted to reduce the test vector data. The algorithm is developed for the test vector compaction and discussed along with results.

KEYWORDS :

Test vector, compaction, ISCAS, ATPG.

1. INTRODUCTION

Test vector size is the big issue in the today's technology. As size of circuit increases the size of test vector also increases so that the memory. This paper includes such ATPG which itself compacts test patterns before further compression of the test patterns. ATPG (acronym for both Automatic Test Pattern Generation and Automatic Test Pattern Generator) is an electronic design automation method/technology used to find an input (or test) sequence that, when applied to a digital circuit, enables testers to distinguish between the correct circuit behavior and the faulty circuit behavior caused by defects[1]. The generated patterns are used to test semiconductor devices after manufacture, and in some cases to assist with determining the cause of failure (failure analysis) the effectiveness of ATPG is measured by the amount of modeled defects, or fault models, that are detected and the number of generated patterns. These metrics generally indicate test quality (higher with more fault detections) and test application time (higher with more patterns). ATPG efficiency is another important consideration. It is influenced by the fault model under consideration, the type of circuit under test (full scan, synchronous sequential, or asynchronous sequential), the level of abstraction used to represent the circuit under test (gate, register-transistor, switch), and the required test quality [2].

The single stuck-at-fault model has been widely accepted as a standard target model to generate a set of test patterns to detect all the stuck faults in the circuit. A single stuck-at fault represents a line in the circuit that is fixed to logic value 0 or 1. The single-stuck fault model is also referred to as the classical or standard fault model because it has been the first and the most widely studied and used. Although its validity is not universal, its usefulness results from the following attributes:

The single stuck at fault can be used to represent short or open ,caused due to short between ground or power line, causing a signal line remain at a fixed voltage level.If we consider single stuck at fault then the number of faults is 2n, where n is number of net .In this case we have to find 2n test vectors, for each fault (stuck at 0, stuck at 1) on each net. Size of test vector becomes large for large combinational circuits. ATE (Automatic Test Equipment) bandwidth problem cause to handle these test vectors and testing time may be more in this case. No doubt test vector compression methods are available, prior to that, befor test generation the number of faults can be reduced sothat the test vectors. The number of faults can be reduced using fault equivalence method and fault dominance method. In this paper an attempt is made to reduce the fault set using fault equivalence method and developed in C++. The logic and flow chart of the program are given in this paper. The results for ISCAS C17 benchmark circuit were analyzed.

The generated test pattern for ISCAS C17 circuit is discussed in result section .The logic and flow chart is discussed in this paper. The test pattern compaction algorithm is discussed along with the results and comparisons.

.

2. ATPG

A defect is an error introduced into a device during the manufacturing process. A fault model is a mathematical description of how a defect alters design behavior. A fault is said to be detected by a test pattern if, when applying the pattern to the design, any logic value observed at one or more of the circuit's primary outputs differs between the original design and the design with the fault. The ATPG process for a targeted fault consists of two phases: fault activation and fault propagation. Fault activation establishes a signal value at the fault model site that is opposite of the value produced by the fault model. Fault propagation moves the resulting signal value, or fault effect, forward by sensitizing a path from the fault site to a primary output [9].

The ATPG process for a targeted fault consists of two phases: fault activation and fault propagation. Fault activation establishes a signal value at the fault model site that is opposite of the value produced by the fault model. Fault propagation moves the resulting signal value, or fault effect, forward by sensitizing a path from the fault site to a primary output.

Fault Activation means to set primary input PI values such that it causes the line having the fault v to the value v'. This is an instance of the line justification problem which deals with finding an assignment of PI values those results in a desired value setting on a specified line in the circuit. Where as the term Fault Propagation means to make the primary output bear value such as the fault is on that particular point itself.

The program for ATPG was developed in C++ language. the flow chart is shown in figure 1.Here we scanned 3 netlist, one for the circuit for which test patterns are to be find, second output of testability measures sothat we can decide which input to be justify using controllability values and which path is to select for error propagation using observability values.

Line justification function shown by part "J" in figure1 was developed separately. Flow chart of Justification function is shown in figure 2. The selection of any one input was done based on controllability 0 and 1 functions. The CC0 (combinational controllability 0) and CC1 (combinational controllability 1) values for both the fanins were identified. If the value to justify was 0, compared both the fanins for CC0 and selected the fan in with minimum CC0.If value to be justify was 1, compared both the fanins for CC1 and selected the fanin with minimum CC1.

Error propagation function is shown in figure 3. If given net was not a primary output but it was a stem then propagation forwarded to any of its branches. Selection of branch was on the basis of observability function. The branches of given stem and index for each branch was identified. The observability for each branch from corresponding element of obs array was identified. The branch with minimum observability was selected and the propagation function was called for selected branch with value for propagation error value (stuck value).We used the output of testability measures program to get the values of controllability 0-CC0, Controllability 1-CC1 and observability. Testability measures helped us to select a line for justification and to select a path for error propagation .Hence this ATPG was based on controllability and observability.

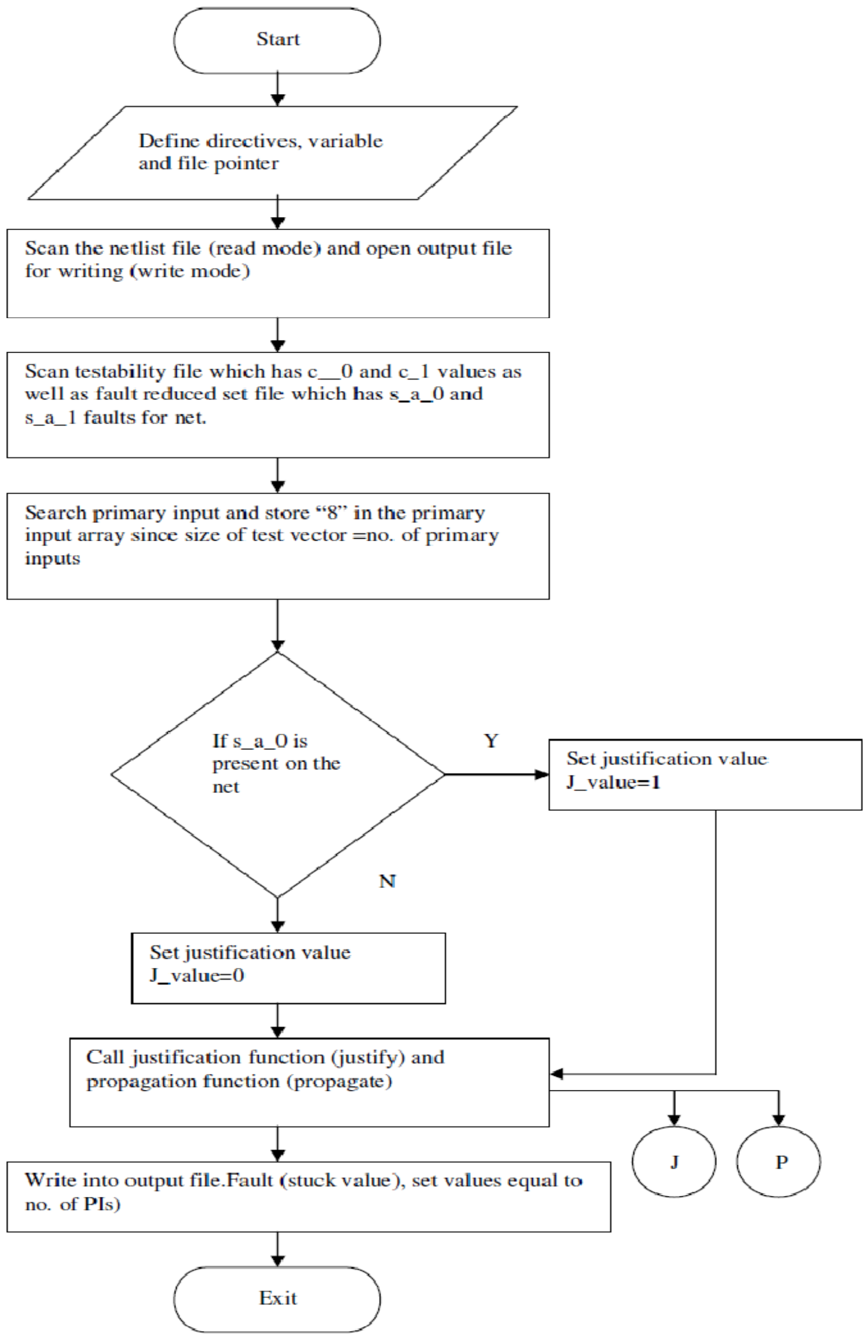

Start
Define directives, variable and file pointer
Scan the netlist file (read mode) and open output file for writing (write mode)
Scan testability file which has c__0 and c_1 values as well as fault reduced set file which has s_a_0 and s_a_1 faults for net.
Search primary input and store “8” in the primary input array since size of test vector =no. of primary inputs
If s_a_0 is present on the net
Y
Set justification value J_value=1
N
Set justification value J_value=0
Call justification function (justify) and propagation function (propagate)
J
P
Write into output file.Fault (stuck value), set values equal to no. of PIs)
Exit

Figure 1. Flow chart of ATPG

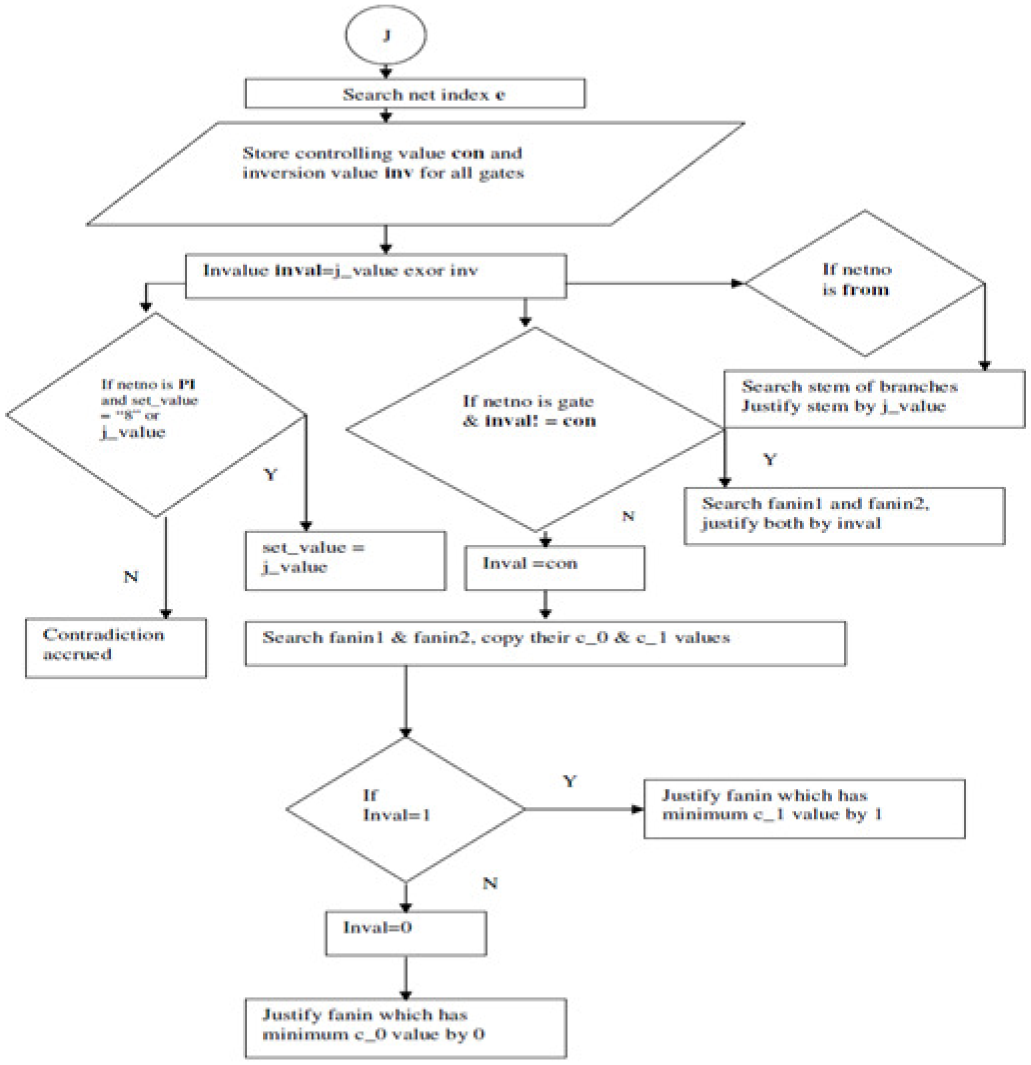

J
Search net index e
Store controlling value con and inversion value inv for all gates
Invalue inval=j_value exor inv
If netno is from
Search stem of branches Justify stem by j_value
If netno is PI and set_value = "8" or j_value
Y
N
set_value = j_value
Contradiction accrued
If netno is gate & inval! = con
Y
N
Search fanin1 and fanin2, justify both by inval
Inval =con
Search fanin1 & fanin2, copy their c_0 & c_1 values
If Inval=1
Y
Justify fanin which has minimum c_1 value by 1
N
Inval=0
Justify fanin which has minimum c_0 value by 0

Figure 2. Flow chart of Line justification

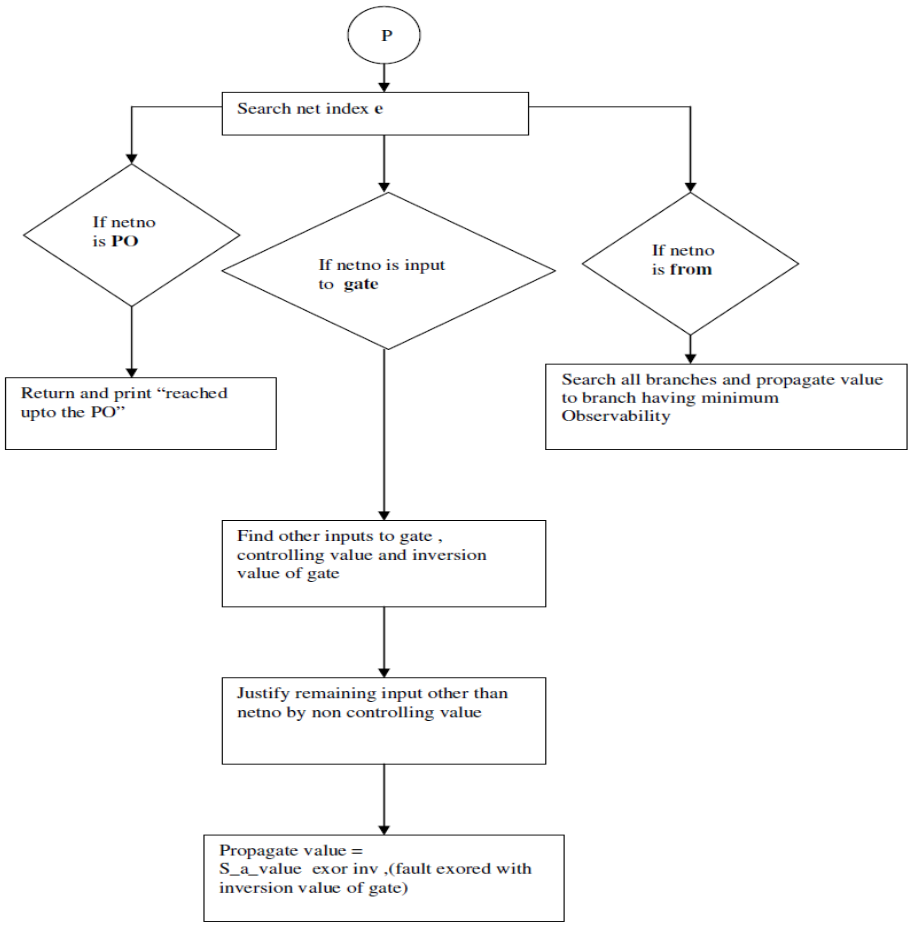

P
Search net index e
If netno is PO
If netno is input to gate
If netno is from
Return and print "reached upto the PO"
Search all branches and propagate value to branch having minimum Observability
Find other inputs to gate , controlling value and inversion value of gate
Justify remaining input other than netno by non controlling value
Propagate value = S_a_value exor inv ,(fault exored with inversion value of gate)

Figure 3. Flow chart of error propagation

3. TEST VECTOR COMPACTION

Amount of Data required to test ICs is growing rapidly in each new generation of technology. Increasing integration density results in larger designs with more scan cells and more faults. Moreover, achieving high-test quality in ever-smaller geometries requires more test patterns. The test vectors generated by ATPG discussed in chapter 6 and 7 can be compress. In this chapter the method to compress the test vector is discussed.

As the complexity of very large scale integration (VLSI) circuit increases, testing plays an important role in today's system design. One of the most important factors in driving up the test cost is increasing the amount of test data volume, which is a result of the large size of the designs and the new types of defects appearing in the advanced manufacturing process. A large amount of test data must be stored in the automatic test equipment (ATE) and transferred deep into the chip as fast as possible. Since the channel capacity and the size of memory of ATE are limited, the test application time and the test power have been significantly increased.

Automatic Test Equipment (ATE) has limited speed, memory, and I/O channels. The test data bandwidth between the tester and the chip, as shown in **figure 1** is relatively low and generally is a bottleneck with regards to how fast a chip can be tested. The chip cannot be tested any faster then the amount of required transferring the test data, which is, equals to:

(Amount of test data on tester)/(number of tester channels x tester clock rate)

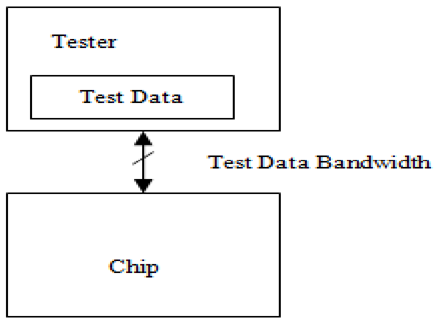


Figure.4 Block diagram Illustrating Test Data Bandwidth

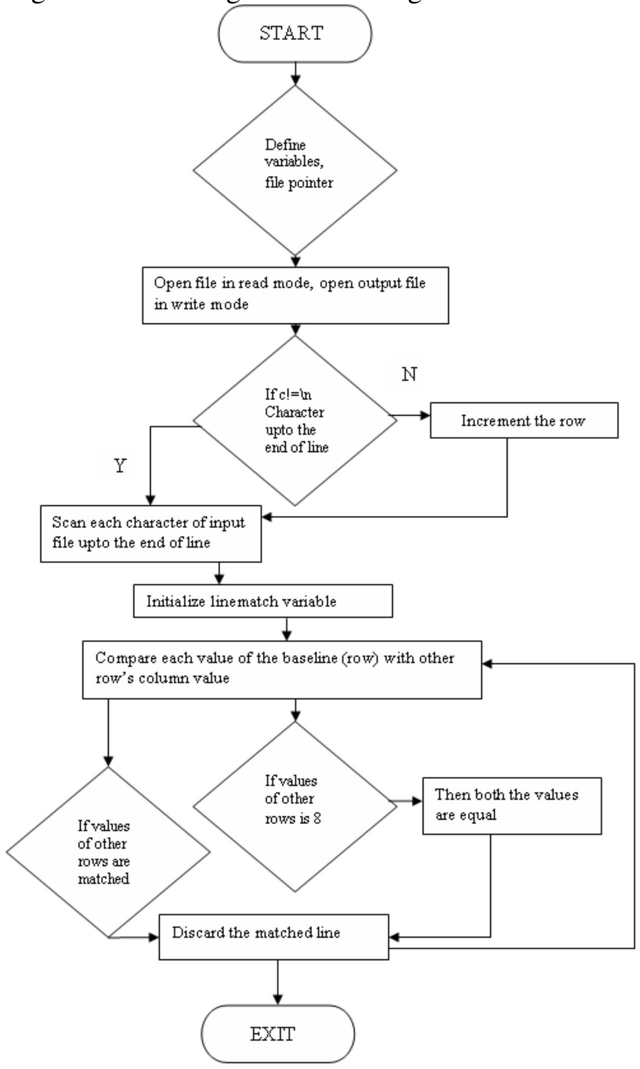

START
Define variables, file pointer
Open file in read mode, open output file in write mode
If c!=\n Character upto the end of line
N
Increment the row
Y
Scan each character of input file upto the end of line
Initialize linematch variable
Compare each value of the baseline (row) with other row's column value
If values of other rows is 8
Then both the values are equal
If values of other rows are matched
Discard the matched line
EXIT

Figure 5. Flow chart for partially specified test pattern compaction

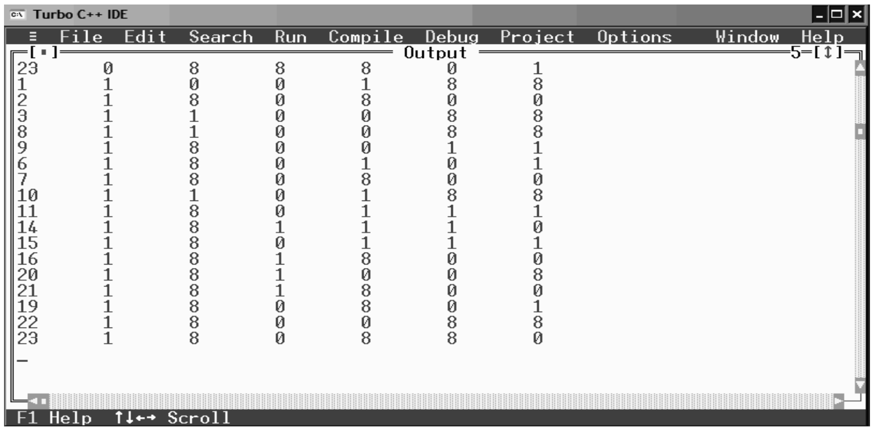
Variables and file pointers were defined. File was open in read mode “r” for which test vectors to be compressed and other for storing results in write mode “w”. while loop was used which scan the variable of file (file for which test vector is to be compress) upto the end of file .If variable c (character) is not equal to end of the line then scan each character of row one by one, else if c=end of the line then increment row and scan the character one by one upto the last column. File scanning, character by character was completed. Linematch variable was initialized to 0.

x is the variable for base line. Base line is the row for which we are comparing other rows. If it is base line (take one variable i, if i! =x) then for all column compare values of base line column one by one. If both the rows (baseline row and any other row) matches then discard the linemathed row .Store its net no and stuck at fault. In the comparing of the column values, if baseline contains 8 (don’t cares X) then the other respective column should contain same value 8.In this loop we had find same test vector. We were not bother about don’t care values. This was solved in next loop. Increment the row; now the base line is different do same as above. While doing so the one matched line was discarded .Then decrement the row size by 1.

If base line column contains “0” or “1” and other line’s corresponding column contains “8” then column value matches. Since 8 may be either “0” or “1”.The matched lines along with their faults were printed in the output file.

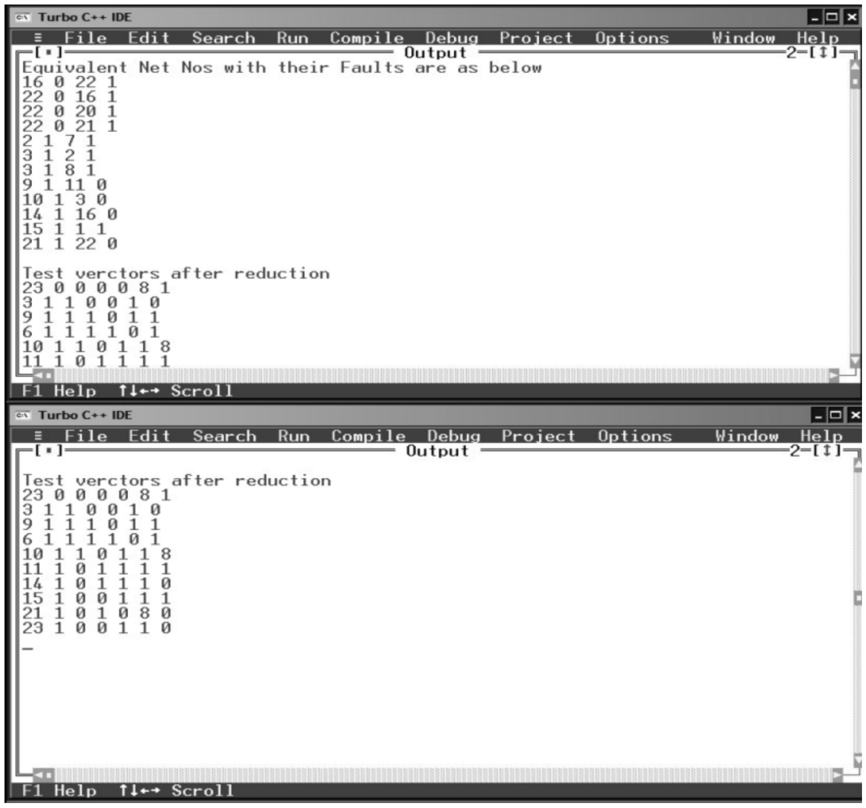
4. RESULTS

The programs for all above concepts were developed in C++.The snap shots of output window are shown in this section for all programs. The programs are generic can be used for any combinational circuit but for the result purpose only results for ISCAS85 C17 benchmark circuit is shown. Figure 6 gives generated test vectors by ATPG based on fault equivalence and testability measures. The length of test vector is equal to the number of primary inputs so that the array size is equal to primary inputs. All other columns represents test vector for corresponding fault on net. “8” was stored instead of don’t care (x) so that array type can not differ. Equivalent net numbers and their corresponding faults are shown in figure 6.


Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
Output
F1 Help ↑↓←→ Scroll

```
23      0     8     8     8     0     1
1       1     0     0     1     8     8
2       1     8     0     8     0     0
3       1     1     0     0     8     8
8       1     1     0     0     8     8
9       1     8     0     0     1     1
6       1     8     0     1     0     1
7       1     8     0     8     0     0
10      1     1     0     1     8     8
11      1     8     0     1     1     1
14      1     8     1     1     1     0
15      1     8     0     1     1     1
16      1     8     1     8     0     0
20      1     8     1     0     0     8
21      1     8     1     8     0     0
19      1     8     0     8     0     1
22      1     8     0     0     8     8
23      1     8     0     8     8     0
_
```

Figure 6. Partially specified test patterns ISCAS 85 C17 benchmark circuit


Turbo C++ IDE
File Edit Search Run Compile Debug Project Options Window Help
Output
F1 Help Scroll

```
Equivalent Net Nos with their Faults are as below
16 0 22 1
22 0 16 1
22 0 20 1
22 0 21 1
2 1 7 1
3 1 2 1
3 1 8 1
9 1 11 0
10 1 3 0
14 1 16 0
15 1 1 1
21 1 22 0

Test verctors after reduction
23 0 0 0 0 8 1
3 1 1 0 0 1 0
9 1 1 1 0 1 1
6 1 1 1 1 0 1
10 1 1 0 1 1 8
11 1 0 1 1 1 1
```

```
Test verctors after reduction
23 0 0 0 0 8 1
3 1 1 0 0 1 0
9 1 1 1 0 1 1
6 1 1 1 1 0 1
10 1 1 0 1 1 8
11 1 0 1 1 1 1
14 1 0 1 1 1 0
15 1 0 0 1 1 1
21 1 0 1 0 8 0
23 1 0 0 1 1 0
_
```

Figure 7. Snap shot of result of test compression for C17 circuit (netlist3)

In figure 6, the first column indicates the net number, second column indicates the fault on corresponding net for example 23 – 0 indicate stuck at 0 fault on net number 23.Other columns indicates the test vector i.e. 88801 or xxx01 is the test vector which should be applied to primary inputs to detect stuck at 0 fault on net number 23.

.

The complete test set after reduction is shown along with net nos. and their faults in figure 7.First window of figure 7 shows the equivalent faults. Stuck at 0 faults on net number 16 is equivalent to stuck at 1 fault on net number 22.So only one test vector was considered.

The table 1 shows the result analysis in terms of bits. It shows the total number of bits before fault equivalence method. Using the fault equivalence method the number of faults get reduced sothat the test vectors .The percentage reduction in bits using fault equivalence is shown. The number of bits is quit reduced using test vector compression method. The percentage reduction using this method for different circuits is given in the table 1.The graphical representation of these results is shown in figure 8.

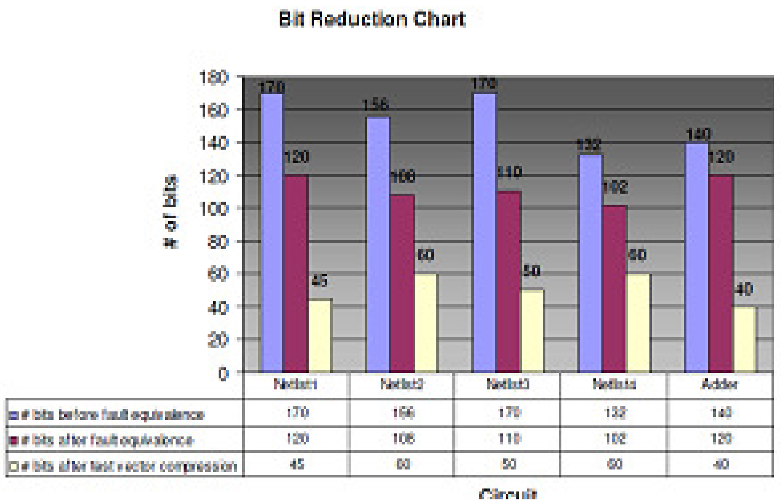


Figure 8.Graphical representation for reduction in # of bits

Circuit	# bits before fault equiva -lence	# bits after fault equiva -lence	% Reduction after Fault Equivalence	# bits after test vector compress-ion	% Reduction after Compress-ion
Netlist 1	170	120	29.41	45	73.53
Netlist 2	156	108	30.77	60	61.54
Netlist 3	170	110	35.29	50	70.59
Netlist 4	132	102	22.73	60	54.55

Table 1. Reduction in # of bits using test vector compression

The table 1 shows the result analysis in terms of bits. It shows the total number of bits before fault equivalence method. Using the fault equivalence method the number of faults gets reduced sothat the test vectors. The percentage reduction in bits using fault equivalence is shown. The percentage reduction using this method for different circuits is given in the table 1.The number of reduction in bits after compaction is also shown in table 1.

5. CONCLUSION

The object oriented generic program was developed to reduce the number of single stuck at faults using fault equivalence method. For these reduced faults, test vectors were generated by controllability and observability aided ATPG based on line justification and error propagation. The test vector compaction algorithm was developed to reduce test vector data. ISCAS 85 C17 benchmark circuit was analyzed along with other circuit netlist for all above concepts

REFERENCES

[1] D. Whitley Sokolov, A. Sanyal and Y. Malaiya. Dynamic power minimization during combinational circuit test as a travelling salesman problem. In the proceeding of the IEEE Congress on Evolutionary Computation, vol.2, pp. 1088-1095,Sept 2005.

[2] Jennifer Dworak, Michael R. Grimaila, Sooryong Lee, Brad Cobb and M. Ray Mercer. A new atpg algorithm to limit test set size and achieve multiple detections of all faults. In the proceedings of Design, Automation and Test, Europe, 2002.

[3] Nur A. Touba. Survey of Test vector Compression Techniques. In the IEEE Design and Test of Computers, July-August 2006.

[4] S.C. Seth, L. Pan and V.D Agrawal, PREDICT: Probabilistic Estimation of Digital Circuit Testability. In the Proceedings of Fault Tolerant Computing System, pp. 220-225, 1985.

[5] S.C. Chang, W.B. Jone and SS. Chang. TAIR: Testability Analysis by Implication Reasoning. In the IEEE Transaction on CAD, Volume 19, No. 1, pp. 152-160, January 2000.

[6] L.H. Goldstein. Controllability/Observability Analysis of digital circuits. In the IEEE Transactions on Circuits and Systems, Vol. 26, pp. 685-693, September 1984.

[7] David Bryan. ISCAS'85 benchmark circuits and netlist format. North Carolina State University.

[8] M. Abramovici and R. Aitken. Error, Fault and Defect Diagnosis: A Detective Story. Tutorial, IEEE International Test Conference, 1997.

[9] Abramovici, Melvin A. Breuer, and Arthur D. Friedman. Digital Systems Testing and Testable Design. Jaico Publication.

[10] Michael Bushnell and Vishwani Agrawal. Essentials of Electronic Testing. Springer Publication.

[11] Xiaoqing Wen Laung Terng Wang, Cheng Wen Wu. VLSI Test Principles and Architectures. Morgan Kaufmann, 2006.

[12] Vaishali H. Dhare, Usha Mehta, "Object Oriented Implementation of Combinational Controllability and Observability Algorithms", International Journal on Electronics Engineering: Kurukshetra, Hariyana Volume2, Number-1, June 2010, Pg.No 93-97.

[13] Vaishali H. Dhare, Usha Mehta, "Development of Controllability Observability Aided Combinational ATPG with Fault Reduction", First International Workshop on VLSI Design 2010 , Chennai , July2010, Pg. No 682-692