Implementation of Algorithm For Testability Measures Using MATLAB

Implementation of Algorithm For Testability Measures Using MATLAB

Prof. Usha Mehta * Dr. N. M. Devashrayee** Dr. K. S. Dasgupta***

Research Scolar (VLSI Design), Nirma University, Ahmedabad : ushasmehta2002@yahoo.co.in, usm_ec.it@nirmauni.ac.in
**PG Coordinator (VLSI Design), Nirma University, Ahmedabad Email : phy_nmd@nirmauni.ac.in
*** Deputy Director, SAC, ISRO, Ahmedabad Email : ksd@sca.isro.org.gov

***Abstract*— Increase in transistor density has a great impact on testing as well as design. In worst case, test complexity increases exponentially with number of transistors and number of flipflops. The amount of data and complexity of data generation required to test ICs is growing rapidly in each new generation of technology. The testing of any circuit is based on two specific tasks : to set a particular net to a specific value (to control the net) and to observe the value available on a particular net(to observe the net). Depending upon the type of net, the types of components connected to the net and the level of the net, the effort needed to control the net or to observe the net varies. In such era, the search process of any Test Generation algorithms involved with the combinational automatic test pattern generation involves two important decisions. The first one being to select one of the several unsolved problems existing at a certain stage in the execution of the algorithm . The second type is to select one possible way to solve the selected problem. Selection criteria differ mainly by the cost functions they are used to measure "difficulty'. This paper describes a technique to estimate the testability of the nets of a combinational circuit, whose description can be either at the gate-level. Three weight functions, CC0(combinational controllability 0), CC1(combinational controllability 1) and CO (combinational observability) are evaluated for each net of the network, and the results obtained are employed in an existing ATPG procedures in order to reduce the test generation time. Experimental results obtained on the ISCAS' 85 benchmark circuits show the effectiveness of the method.**

I. INTRODUCTION

The study and the implementation of efficient algorithms for integrated circuits testing becomes every day more difficult because of the large number of logic gates that can be packed together on a single silicon wafer [1,2].

An automatic test pattern generation (ATPG) program is a computer aided design (CAD) tool that, given a fault in the circuit under test, calculates the sequence of input vectors (i.e. the test pattern) that has to be applied to the circuit in order to detect the fault. Given the location of the fault, the procedure starts investigating all possible paths of the network that allow its propagation through the circuit, until its symptoms can be observed on a primary output (forward propagation); then, it computes the input vectors required to satisfy all the conditions imposed on the internal nodes of the network during the propagation phase (backward justification) [3]. It can be observed that the number of possible paths that the test generator must examine, in order to compute a test pattern, increases exponentially with the number of nodes of the circuit, i.e. the number of gates [4,5].

This paper describes a procedure that can be used to estimate the testability of the nodes of a combinational circuit [6-81; an existing ATPG procedure [9] uses the algorithm as a pre-processing step in order to improve its performance; in fact, as proved in [2], searching test patterns using the nodes of the circuit having the highest testability insures the reduction of the test generation time, and guarantees the computation of a better quality test set.

The rest of this paper is organized as follows. Section 2 describes the motivation behind developing testing measures algorithm, Section 3 introduces the circuit representation that has been used. Section 4 formally defines the testability measures, section 5 gives the implementation steps for their computation. Section 6 illustrates how these measures can be determined on a practical example. Section 7 shows the benefits, in terms of test generation time, produced by the application of the testability measures to the test generator described in [9]. Finally, Section 8 is devoted to conclusions.

II MOTIVATION

The amount of data and complexity of data generation required to test ICs is growing rapidly in each new generation of technology. The combinational automatic test pattern generator (ATPG) is the main building block of any testing tool. The automatic test pattern generator for combinational design is of the most importance because after scan chain insertion, any of sequential testing will also use the combinational ATPG for a given state of sequential circuit. So it may be said that the combinational ATPG is the backbone of today's testing tool. The testing of any circuit is based on two specific tasks. 1. To set a particular net (either primary input or any internal net to a particular value) 2. To observe the value on a particular net (i.e. either primary output or any internal net). Depending upon the type of net, the types of components connected to the net and the level of the net (i.e place : how far it is from the primary input or output) the effort needed to set that net or to observe that net varies. In such era, the search process of any Test Generation(TG) algorithms involved with the combinational ATPG involves two important decisions. The first one being to select one of the several unsolved problems existing at a certain stage in the execution of the algorithm . The second type is to select one possible way to solve the selected problem. Selection criteria differ mainly by the cost functions they are used to measure "difficulty'. Typically cost functions are of two types:

1. Controllability - For a digital circuit it is defined as the difficulty of setting a particular logic signal to state 0 or 1.
2. Observability - For a digital circuit it is defined as the difficulty of observing the state of a logic signal .

Controllability measures can be used both to select the most difficult line-justification problem , and then to select among the unspecified inputs of G the one that it is easiest to set to the controlling value of the gate. Observability measures can be used to select the gate from the D-frontier whose input error is the easiest to observe. Goldstein was the first one to implement a computer program to calculate those controllability and observability values which he named as SCOAP acronym for Sandia Controllability and Observability Analysis Program.

The testing system consists of two basic tasks 1. Test Generation 2. Fault Simulation.
To reduce the cost and complexity of test pattern generation, the following goals should be achieved.

1. to increase the fault coverage
2. to reduce the overall efforts (CPU time)
3. to reduce the number of test vectors required (to reduce the test application time)

To reduce the cost and complexity of fault simulation, the following goals should be achieved

1. to reduce the fault list
2. to use efficient and diverse fault simulation methods
3. to use efficienf fault sampling methods

The testability measures is a powerful heuristic used during test generation which will show its effect on fault simulation also.
The testability measures are useful in following ways.

- In ATPG during *backtracing* (controllabilities) and *propagating* the fault effect (observabilites) since they give the path of least resistance. Some caution is necessary since reconvergent fanout introduce error in controllabilites and fanout stems introduce error in observabilities.
- It tells designer which parts of the design are extremely hard-to-test. Either redesign or special-purpose test hardware is needed to achieve high fault coverage.
- Extremely useful for estimating fault coverage and test vector length. Fault coverage estimation via testability can reduce CPU time by orders of magnitude over fault simulation (and has only a 3-5% error).

III CIRCUIT FORMAT

In this section we discuss the format of the circuit specification that we input to the program which calculates the testability measures of the circuit. In particular, we focus our attention on circuit description
The topological description of the combinational network under consideration is ***gate –level.***
Any ATPG tool takes the Design Under Test (DUT) as an input and generate the test vector for given fault in the DUT. Most of the ATPG tool accepts DUT as an input in form of netlist (text file) and by processing on netlist; it generates the fault directory which will contain the fault and its corresponding test vector. Considering the same concept, for testability algorithm implementation also the input is required in the standard format for netlist. A set of standard benchmark type circuit set on which the results can be proved was also required. For both of these reasons, ISCAS 85 (International Symposium for Circuits and Systems) combinational circuits and netlist format were quite adequate so the ISCAS85 format is used as an input netlist format.

IV TESTABILITY MEASURES

An attempt to *quantify* testability by Goldstein '79 and Grason '79 resulted in two testability measures, controllability and observability. Controllability is defined as the difficulty of setting a particular logic signal to a 0 or a 1. PIs are free (usually assigned a value of 1). Output 1 values for AND gates are more expensive than OR gates.
Observability defined as the difficulty of observing the state of a logic signal. Purpose to define such terms are :

1. Analysis of difficulty of testing internal circuit nodes
2. May need to modify circuit, add observation points or test hardware
3. Can be used to guide ATPG algorithms, i.e., to help them make decisions by providing information about the difficulty of setting lines.
4. Can be used to estimate fault coverage
5. Can be used to estimate test vector length.

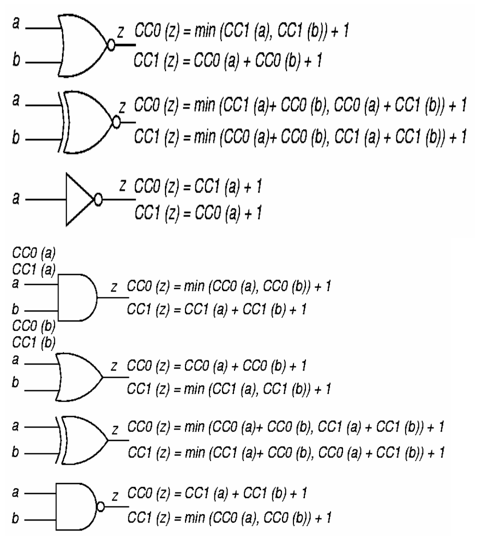
Testability analysis attributes requires topological analysis (but no test vectors). It should be linear in complexity otherwise it's pointless, i.e., might as well use ATPG to compute exact fault coverage.
• *Controllability*: From 1 (easiest) to infinity (hardest).
• *Observability*: From 0 (easiest) to infinity (hardest).
Combinational measures are related to the number of signals that may be manipulated to control or observe *l*.
This method consists of 3 numerical measures for each signal (*l*) in the circuit:
• Combin. *0-controllability*, CC0(*l*)
• Combin. *1-controllability*, CC1(*l*)
• Combin. *observability*, CO(*l*)

Controllabilities:

The basic process: Set PIs to 1, progress from PIs to POs, add 1 to account for logic depth.
In general, if only one input sets gate's output:
output controllability = *min*(input controllabilites) + 1

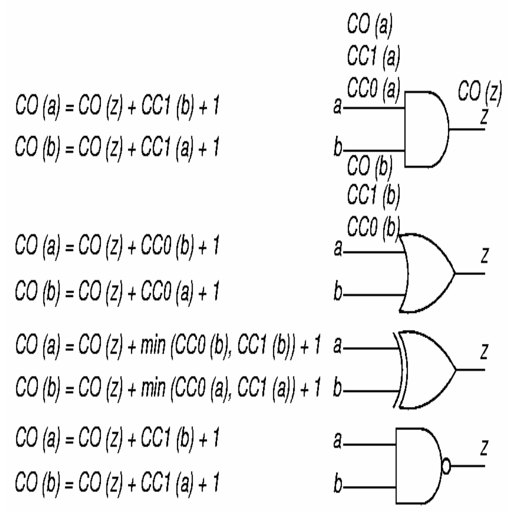
If all inputs set gate output:
output controllability = *sum*(input controllabilities) + 1
If gate output is determined by multiple input sets, e.g., XOR:
output controllability = *min*(controllabilities of input sets) + 1
For the fanouts the controllability values remain to be same as for the line from which it is emerging. Details are given in figure 2.1

Observabilities:
The basic process for observability is started after controllabilities computed, set P0s to 0, progress from PO to PIs, add 1 to account for logic depth.
For example, the difficulty of observing a designated input to a gate is the sum of (1) the output observability (2) the difficulty of setting all other inputs to *non-dominant* values (3) plus 1 for logic depth. Further details are given in figure 2.2

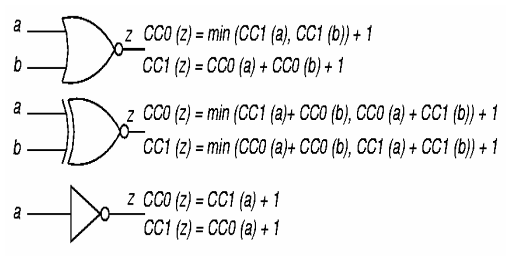


a
b
z
CC0 (z) = min (CC1 (a), CC1 (b)) + 1
CC1 (z) = CC0 (a) + CC0 (b) + 1
CC0 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
CC1 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
CC0 (z) = CC1 (a) + 1
CC1 (z) = CC0 (a) + 1
CC0 (a)
CC1 (a)
CC0 (b)
CC1 (b)
CC0 (z) = min (CC0 (a), CC0 (b)) + 1
CC1 (z) = CC1 (a) + CC1 (b) + 1
CC0 (z) = CC0 (a) + CC0 (b) + 1
CC1 (z) = min (CC1 (a), CC1 (b)) + 1
CC0 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
CC1 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
CC0 (z) = CC1 (a) + CC1 (b) + 1
CC1 (z) = min (CC0 (a), CC0 (b)) + 1

Figure 1 Controllability Measures



CO (a) = CO (z) + CC1 (b) + 1
CO (b) = CO (z) + CC1 (a) + 1
CO (a)
CC1 (a)
CC0 (a)
CO (z)
CO (b)
CC1 (b)
CC0 (b)
a
b
z
CO (a) = CO (z) + CC0 (b) + 1
CO (b) = CO (z) + CC0 (a) + 1
CO (a) = CO (z) + min (CC0 (b), CC1 (b)) + 1
CO (b) = CO (z) + min (CC0 (a), CC1 (a)) + 1
CO (a) = CO (z) + CC1 (b) + 1
CO (b) = CO (z) + CC1 (a) + 1



a
b
z
CC0 (z) = min (CC1 (a), CC1 (b)) + 1
CC1 (z) = CC0 (a) + CC0 (b) + 1
CC0 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
CC1 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
CC0 (z) = CC1 (a) + 1
CC1 (z) = CC0 (a) + 1

Figure 2 Observability Measures

The accuracy problem occurs for the computation of the observability of a fanout stem with *n* branches. One attempt is to bound the stem probability by: • *min*(all fanout branch observabilities) The events of observing a signal through each branch are independent. • *max*(all fanout branch observabilities) They are all dependent, therefore branch that's hardest to observe is correct choice.
For our implementation, the first option is chosen.

V IMPLEMENTATION

This testability_tool accepts the DUT in ISCAS85 netlist format text file and generates the combinational controllability 0(CC0), combinational controllability 1(CC1) and combinational observability (CO) for each of the nets in given circuit. It stores results in controlobserv.txt file which file further can be used as a part of ATPG tool.

The main requirement here was linear computational complexity otherwise it is pointless – might as well use automatic test-pattern generation and calculate exact fault coverage exact test vectors and So the results of controllability are derived by computing first the controllability of each node and then using it, observability is derived so this code is linear in computational complexity.

Following steps describes the algorithm implementation.

1: Open the netlist file which is in text format.
2: Assign each column of netlist to corresponding array
   i) Array 1 : net number
   ii) Array 2 : gat
   iii) Array 3 :type of net (inpt, nor, nand, od, nor, from)
   iv) Array 4 : total fan out from that net
   v) Array 5 : total fan in to that net (if net is gate output then inputs to gate is fan in to that net)
   vi) Array 6 : fan in 1
   vii) Array 7 : fan in 2
3: Initialize the arrays for 1. controllability 0 (c_0) 2. controllability 1 (c_1) 3. Observability (observ) of length equal to total number of net in circuit. All the element of c_0 and c_1 is assigned value 0 and all the elements of observ are assigned with -1. As controllability can be any value out of [1, α) and value of observability can be [0, α), the logic here is

to assign value which is an unvalid or not possible value for that function.

Starting calculation for controllability

4: Set element of c_0 and c_1 corresponding to primary input to value 1

5: Now we will start to count value for other nets by calling controllability fuction for each of the net in array 1.

Controllability function

6: If current net is an input , already c_0 and c_1 is set.

7: If it is not an input, it is output from the gate or branch from a stem

8: If it is branch of the stem, as the branch has the same controllability values as stem, find its stem from array 3 and index of stem in array 1 and corresponding element value from c_0 and c_1. If this values are non zero, assign this value at index for given stem in c_0 and c_1 respectively.

9: If given net is output from some gate, find the type of gate from array 3 (like nor, nand, or, and), number of fan ins to this gate from array 4 and name of fan ins from array 6 and 7(let's call it fan1 and fan2).

10: Find out the index of fan1 and fan2 from aray1 and corresponding element from c_0 and c_1 each for fan1 and fan2. (let's call it c1_0, c1_1, c2_0, c2_2)

11: Apply following rules to calculate the c_0 and c_1 of current net.

12: If c1_0 and c2_0 are non zero,

For 'and'	c_0(e)=min(c1_0, c2_0) +1;
For 'nand'	c_1(e)=min(c1_0, c2_0) +1;
For 'or'	c_0(e)= c1_0 + c2_0 +1;
For 'nor'	c_1(e)= c1_0 + c2_0 +1;

13: If c1_1 and c2_1 are non zero,

For 'and'	c_1(e)= c1_1 + c2_1 +1;
For 'nand'	c_0(e)= c1_1 + c2_1 +1;
For 'or'	c_1(e)= min(c1_1, c2_1) +1;
For 'nor'	c_0(e)= min(c1_1, c2_1) +1;

14: Make the array of elements of c_0 and c_1 with nonzero values. If the length of this array is same as length of total number of net than, all the net is considered for c_0 and c_1. If not then repeat until both arrays have same length.

Observability

15: For elements of array 1 which are primary output, set corresponding element of observ to 0 value.

16: for each of the member of array 1 , call the function observability

Observability function

17: find the index of net for which observability is being calculated

18: If given net is not a primary output or stem, it should be input to a gate. Find out the type of that gate, second input to that gate and fan out net from that gate.

19: Find the index for fan out net from gate, and index for second input to that gate.

20: Find c_o and c_1 for second input to gate using its index.

21: Find the observability for output from gate using its index and observ array.

22: If observability of output from gate is not -1, it is a valid observability. Calculate the observability for net under consideration using following rule.

For 'and' gate : observ(e)= observ(output_index)+ c_1_second_ip +1;
For 'nand' gate : observ(e)= observ(output_index)+ c_1_second_ip +1;
For 'or' gate : observ(e)= observ(output_index)+ c_0_second_ip +1;
For 'nor' gate : observ(e)= observ(output_index)+ c_0_second_ip +1;

23: If given net is a stem, find its branches and corresponding index.

24: Find the observabilities for both the branches

25: Assign minimum observability out of the two observabilities for branches to stem.

26: Make an array from observ which has element values more than -1. If the length of this array is equal to total number of net, all valid observabilities has set else repeat the loop.

VI An application example

As an application example of the proposed technique, let us consider the circuit C17. It is the smallest circuit in the ISCAS *'85* [lo] set, and its schematic is shown in Fig. 3. Notice that the S27 description contains both gate and block elements. By applying the COs and 00s

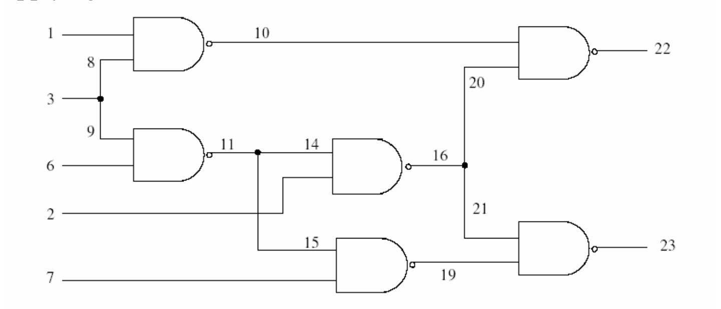


Figure 3 ISCAS'85 Benchmark Circuit C17

CC0(10)=CC1(1)+CC1(8)+1= 3
CC1(10)=min(CC0(1),CC0(8))+1= 2
CC0(11)=CC1(9)+CC1(6)+1= 3
CC1(11)=min(CC0(9),CC0(6))+1= 2
CC0(16)=CC1(9)+CC1(6)+1= 4
CC1(16)=min(CC0(9),CC0(6))+1= 2
CC0(19)=CC1(15)+CC1(7)+1= 4
CC1(19)=min(CC0(15),CC0(7))+1= 2
CC0(22)=CC1(10)+CC1(20)+1= 5
CC1(22)=min(CC0(10),CC0(20))+1= 4
CC0(23)=CC1(21)+CC1(19)+1= 5
CC1(23)=min(CC0(21),CC0(19))+1= 5

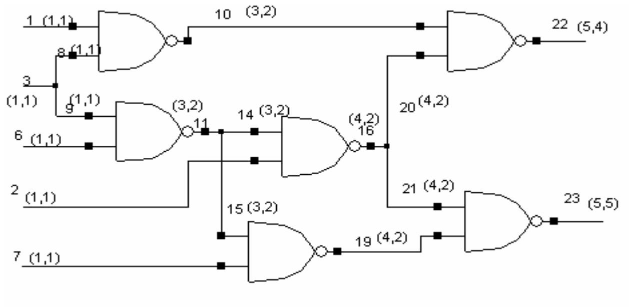


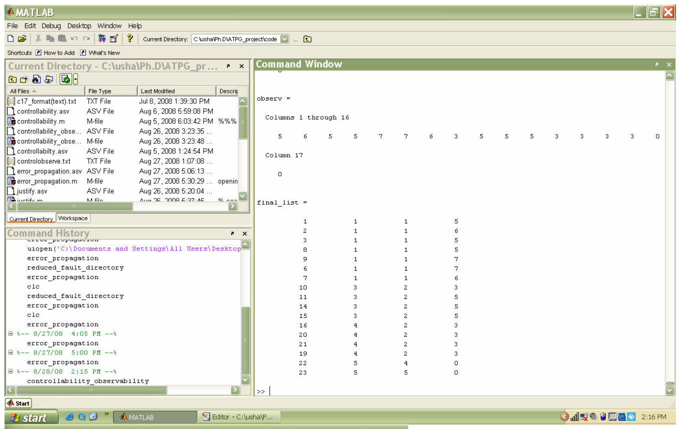
Figure 4 The calculated Controllabilities for each net in C17.

VII Results

The above implementation is done using MATLAB 7.0 software. The implementation is tested using ISCAS85 benchmark circuits and other higher order circuits. The values obtained by tool are perfectly matching with calculated values in all the case.

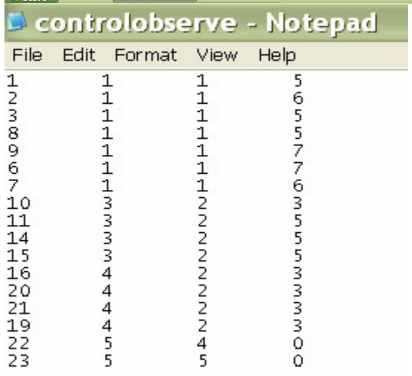
Only two cases are given below for demonstration view.

1. For ISCAC85 C17 circuit



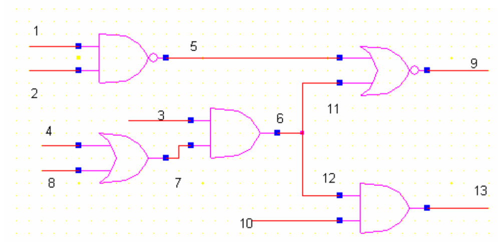
```
observ =

  Columns 1 through 16

     5     6     5     5     7     7     6     3     5     5     5     3     3     3     3     0

  Column 17

     0


final_list =

     1     1     1     5
     2     1     1     6
     3     1     1     5
     8     1     1     5
     9     1     1     7
     6     1     1     7
     7     1     1     6
    10     3     2     3
    11     3     2     5
    14     3     2     5
    15     3     2     5
    16     4     2     3
    20     4     2     3
    21     4     2     3
    19     4     2     3
    22     5     4     0
    23     5     5     0

>>
```

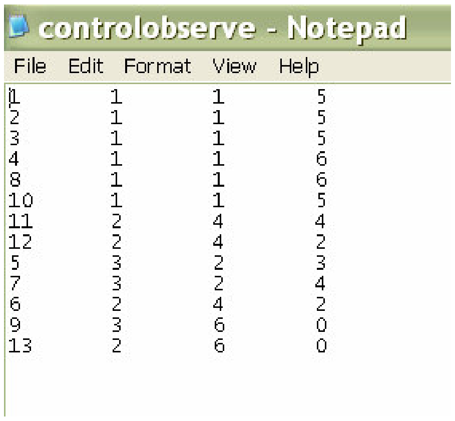
```
uiopen('C:\Documents and Settings\All Users\Desktop
error_propagation
reduced_fault_directory
error_propagation
clc
reduced_fault_directory
error_propagation
clc
error_propagation
%-- 8/27/08  4:05 PM --%
error_propagation
%-- 8/27/08  5:00 PM --%
error_propagation
%-- 8/28/08  2:15 PM --%
controllability_observability
```



controlobserve - Notepad

```
1        1        1        5
2        1        1        6
3        1        1        5
8        1        1        5
9        1        1        7
6        1        1        7
7        1        1        6
10       3        2        3
11       3        2        5
14       3        2        5
15       3        2        5
16       4        2        3
20       4        2        3
21       4        2        3
19       4        2        3
22       5        4        0
23       5        5        0
```





controlobserve - Notepad

```
1        1        1        5
2        1        1        5
3        1        1        5
4        1        1        6
8        1        1        6
10       1        1        5
11       2        4        4
12       2        4        2
5        3        2        3
7        3        2        4
6        2        4        2
9        3        6        0
13       2        6        0
```

VI Future Scope

The assumption of signal independence is the key behind this linear time algorithm. However, this reduces its accuracy in predicting which individual faults will remain *undetected* and which will be detected. So for higher complex circuits, predictive controllability concepts can be used.

VII References

[1] H. Fujiwara, ***Logic Testing and Design for Tesfability.*** MIT Press, Mass. (1985).

[2] M. Abramovici, M. A. Breuer and A. D. Friedman, ***Digital Systems Testing and Testable Design.*** Computer Science Press (1990).

[3] G. R. Putzolu and J. P. Roth, A heuristic algorithm for the testing of asynchronous circuits. IEEE *Trans. Computers C-20, 639-647 (1971).*

[4] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Circuits.* Computer Science Press (1976).

[5] A. Miczo, The sequential ATPG: a theoretical limit. ITC-83: *IEEE International Test Conference,* pp. 143-147 (1983).

[6] L. H. Goldstein, Controllability/observability analysis of digital circuits. ***IEEE Trans. Circuits Systems CAS26, 685493 (1979).***

[7]J. Grason, TMEAS-a testability measurement program. ***DAC-16: 16th IEEE Design Automarion Conference,*** pp. 156-161 (1979).

[8] J. E. Stephenson and J. Grason, A testability measure for register transfer level digital circuits. FTCS-6: ***6th IEEE International Symposium on Fault-Tolerant Compuling,*** pp. 101-107 (1976).

[9] E. Macii and A. R. Meo, Algebraic/topological combined strategies for sequential ATPG. ***25th IEEE AsilomarConference on Signals, Systems and Computers,*** pp. 1240-1244 (1991).

[10] F. Brglez, D. Bryan and K. Kozminski, Combinational profiles of sequential benchmark circuits. ISCAS '89: ***IEEE International Symposium on Circuifs and Systems,*** pp. 1929-1934 (1989).

[11] Micahel L. Bushnell, Vishwani D. Agrawal, *Essentials of electronic testing for digital, memory and mixed signal VLSI circuits*, Kluwer Academics Publishers, 2002.

[12] L.H. Goldstein & Evelyn .L.Thigpen, *SCOAP : SANDIA Controllability/Observability analysis program,*SANDIA national laboratories

[13] L.H. Goldstein, Controllability/Observability analysis of digital circuits, Cande workshop, Mt. Hood Oregon (1978)

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

[15] http://iwls.org/iwls2005/benchmarks.html

[16] http://www.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html

[17] http://www.cerc.utexas.edu/itc99-benchmarks/itsw.bench.pdf