Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 745861, 10 pages
doi:10.1155/2012/745861

Research Article

Test Generation for Crosstalk-Induced Delay Faults in VLSI Circuits Using Modified FAN Algorithm

S. Jayanthy,[1] M. C. Bhuvaneswari,[2] and Keesarapalli Sujitha[3]

[1] *Department of ECE., Sri Ramakrishna Engineering College, Tamil Nadu, Coimbatore 641022, India*
[2] *EEE Department, P.S.G. College of Technology, Tamil Nadu, Coimbatore 641004, India*
[3] *M.E Applied Electronics, P.S.G. College of Technology, Tamil Nadu, Coimbatore 641004, India*

Correspondence should be addressed to S. Jayanthy, sjayanthyabi@gmail.com

Received 1 August 2011; Revised 25 October 2011; Accepted 30 October 2011

Academic Editor: Wolfgang Kunz

As design trends move toward nanometer technology, new problems due to noise effects lead to a decrease in reliability and performance of VLSI circuits. Crosstalk is one such noise effect which affects the timing behaviour of circuits. In this paper, an efficient Automatic Test Pattern Generation (ATPG) method based on a modified Fanout Oriented (FAN) to detect crosstalk-induced delay faults in VLSI circuits is presented. Tests are generated for ISCAS_85 and enhanced scan version of ISCAS_89 benchmark circuits. Experimental results demonstrate that the test program gives better fault coverage, less number of backtracks, and hence reduced test generation time for most of the benchmark circuits when compared to modified Path-Oriented Decision Making (PODEM) based ATPG. The number of transitions is also reduced thus reducing the power dissipation of the circuit.

1. Introduction

As a consequence of technological advances which have resulted in an increase of VLSI chip density, increased number of interconnect layers and in an improvement of timing performances, the test for static stuck-at faults only has turned out to be insufficient, and it is now also required to deal with physical defects which affect the timing behavior of a given circuit. Various noise sources such as crosstalk and power supply noise have a significant impact on the timing performance of Deep Submicron (DSM) designs. The increasing number of transistors in the chip leads to more devices switching simultaneously resulting in power supply noise which reduces device voltage levels and increases signal delay. Interconnection lines which were assumed to be electrically isolated can now interfere with each other leading to functional problems. One such interaction caused by parasitic coupling between wires is known as crosstalk. These noise effects can cause completely validated chip to malfunction and lead to performance degradation of deep submicron design.

There are two main types of crosstalk effects: crosstalk-induced pulses and crosstalk-induced delay. The type of crosstalk effect dealt in this paper is crosstalk-induced delay. Crosstalk delay is induced when two lines, an aggressor line (A-line) and victim line (V-line) have simultaneous or near simultaneous transitions, which may cause undesirable effects including glitches, increase, or decrease in the signal delay [1]. If both the lines transit in the same direction, the effective delay is reduced leading to crosstalk speedup. If aggressor and victim transit in the opposite direction, then there will be an increase in delay leading to crosstalk slowdown.

The designer has two options to eliminate errors caused by crosstalk by resizing drivers, rerouting signals, shielding interconnect lines and other such redesign techniques, or by developing techniques to generate tests for crosstalk.

The latter option is often taken by designers as redesign may be very expensive. Moreover, test generation for crosstalk also enables more aggressive design and enables more comprehensive postmanufacturing testing.

Energy dissipation has become a major concern in today VLSI technology with increasing use of wireless communication and portable computers. The acceptance of this area of communication and products depends on their power of consumption. Further periodic testing of VLSI circuits can

cause generation of excessive heat which can damage the chips under test. During testing the low correlation between test vectors increases the switching activity in the circuit leading to higher power dissipation than during its normal operation.

In CMOS, combinational circuits power dissipation is due the following sources: static power dissipation due to leakage currents and other currents drawn continuously from the power supply, dynamic power dissipation due to switching transient current, and charging and discharging of load capacitances [2]. Unlike bipolar technologies, where a majority of power dissipation is static, the bulk of power dissipation in properly designed existing CMOS digital circuits is due to dynamic power dissipation caused by charging and discharging of capacitances. Thus, a majority of the low power design methodology is dedicated to reducing this predominant factor of power dissipation.

However, there are also other components of power dissipation in CMOS circuits like short circuit current power, leakage current power, and static-biasing power. These are negligible when compared with the dynamic power dissipation.

The consumed energy directly corresponds to the switching activity generated in the circuit during test application.

In this paper, a test generation algorithm using a modified version of FAN (fanout-oriented test generation) algorithm for crosstalk delay faults is presented. The test generator detects crosstalk faults that produce delay affect at the primary output. This algorithm also reduces the number of transitions when compared to PODEM algorithm and thus reduces the power dissipation in CMOS circuits.

The remainder of the paper is organized as follows.

Section 2 discusses prior work. Section 3 describes the algorithm to find the paths in the circuit. Section 4 describes the signal values used in test generation test generation. Section 5 describes the test generation algorithm using a modified version of FAN algorithm, and Section 6 presents the experimental results for ISCAS'85 and enhanced scan version of ISCAS'89 Benchmark circuits. The paper is concluded in Section 7.

2. Prior Work

Rubio et al. [3] propose a methodology which is based on a search for a two-vector input pattern that forces a determinate value to the affected nodes and provokes a transition of the affecting lines allowing the propagation of the possible noise effect to the primary output nodes. The relationship of transition propagation in logic circuits has been used to generate the test patterns. Their paper deals with crosstalk pulses.

Chen et al. [4–6] have given a mixed signal test generator XGEN for crosstalk-induced delay faults. They proposed a mixed signal test generation process where characteristics of crosstalk-induced noise are accurately modeled. By using Laplace transformations, they have obtained an expression for crosstalk in the s-domain, which they have transformed back to the time domain. These expressions are used to quantify the dependence of the pulse attributes on the lumped circuit parameters and the rise time of the input transition. Static timing analysis provides timing windows at gate inputs and outputs. The target-timing window is the intersection of the aggressor and victim timing window. For a specific target coupling fault, a pair of vectors create a crosstalk effect at the target and either a logic error or delay at the primary output. Their ATPG algorithm uses 11 valued algebra, analog delays modified PODEM for backtrace procedure. Their algorithm is not complete because of restricted propagation conditions of fault effects, and a constrained logic value system is used.

Krstic et al. [7] developed a constrained path delay model as a combination of a critical path and a set of crosstalk noise sources interacting with the path. It uses a conventional path delay ATPG process without justification to sensitize the fault. Then a genetic algorithm is used to deal with timing information and justification of effecting transitions to primary inputs. Their technique was time consuming since it was based on the genetic algorithm and did not deal with the timing information efficiently.

Li et al. [8] and Shen et al. [9] have proposed a test generation technique based on a single precise crosstalk-induced path delay fault model. However, this approach only generates patterns for a single aggressor affecting a target path. So it cannot propagate a maximal crosstalk-induced effect on a critical path. Further, their algorithm is computationally complex.

In [10], Bai et al. have proposed a solution for multiple aggressor crosstalk problems. In their work, an implication graph is constructed that consists of logic variables and structural information to check for logic conflicts. A modified version of PODEM algorithm is used to search for test vectors.

Aniket and Arunachalam [11] proposed an algorithm for testing crosstalk-induced delay faults. Their algorithm generates a list of critical paths by static timing analysis of the circuits. A robust testability criterion is applied to check for sensibility of the paths. For a sensitizable path the associated aggressors—victim pairs—are activated in a manner that will maximize the aggressor influence on the path to induce maximum crosstalk slowdown along a path. But their technique resulted in greater CPU time.

Sinha et al. [12] have proposed a test generator by deriving a new 57-valued algebra and modified the key ATPG procedures to obtain a test generation methodology that increases the fault coverage, prove more faults as being untestable, and search much larger fraction of the space of all vectors. Their proposed algebra considers single element values called *basis values*, as well as many composite values made of multiple basis values. The assignment of a composite value to a circuit node indicates that any of the basis values comprising the composite value is possible. The use of composite values allows their ATPG method to postpone making a decision until one is absolutely necessary. This makes the search for a test more efficient as it reduces backtracking by reducing the number of assignments that causes conflicts. But the disadvantage is the test generation time is increased. Further, their algorithms considered only

a single pair of a victim line and an aggressor line and hence cannot propagate a maximal crosstalk-induced effect along a critical victim path, on which the effect is more likely to cause delay test failure.

Chun et al. [13] have proposed a test-generation method for crosstalk-induced delay effects, where they have considered multiple-aggressor crosstalk faults to maximize the noise of the victim line. The proposed algorithm uses parasitic information, such as coupling capacitance between a node of the victim and an aggressor, and timing information, such as static timing window and the crosstalk-induced noise delay model. Using the parasitic and timing information, the ATPG can reduce many false aggressors and handle multiple possible aggressors coupled to a victim lying along a path. Spatial pruning is used to remove false aggressors. Static timing analysis is performed then to remove false crosstalk faults. Their ATPG uses the do not care values in the test patterns obtained from the conventional path-delay ATPG to find the worst-possible-case test patterns for the generated crosstalk, and hence their ATPG can reduce the search space of the backward implication of the aggressor's constraints, thus reducing time.

In this paper we have used a modified version of FAN [14] algorithm for generating tests for crosstalk delay faults. FAN is more efficient than PODEM [15], which has been used in the previous papers because there are two extensions to the backtracking concept, namely, backtracking stops at internal fanout lines and FAN satisfies multiple objectives. Hence test generation will be relatively faster compared to PODEM-based test generation. Further, in this paper we have increased the propagations conditions which can increase the fault coverage. To propagate a to-controlling delay effect at the input of a gate, any of three values are allowed on the side input, namely a static noncontrolling, a to-controlling transition and hazardous noncontrolling transition.

3. Crosstalk Fault List Generator

The total numbers of crosstalk faults are very large and impractical to test. Moreover, all the crosstalk faults need not be tested and cannot be tested. The target crosstalk faults are identified by using both the topological and timing information. The steps for finding the target crosstalk faults are given below.

The method uses logical level implementation of the circuit and does not require layout information [16]. It is more amenable to the time to market need of designs. In order to obtain a reduced list of crosstalk delay faults classes of false crosstalk faults that need not be tested has to be identified and should not be included in the target fault list.

The conditions for victim and aggressor to from a crosstalk target fault are:

(1) the victims should lie in the longest path;

(2) the relationship between transition time at the victim line be T_{vic} and transition time at the aggressor line T_{agg} should satisfy the inequality $T_{\text{vic}} - \delta \leq T_{\text{agg}} \leq T_{\text{vic}} + \delta$, where δ can be 1- or 2-unit delay. The value of δ is taken as 1-unit delay. The transitions are limited to timing window δ to cause slowdown on the V-line.

The algorithm to find the longest paths, victim set, and reduced set of target crosstalk delay faults is described.

(1) For timing information, the latest transition time and earliest transition time at each line are calculated. Latest transition time at a line is the maximum delay on any path from any primary input. The latest transition time is calculated using the following rules:

(i) if Line L is a primary input or pseudoprimary input:
latest transition time for L = 1 unit delay;

(ii) if L is an output of a gate G having n inputs: $I_1, I_2, I_3, \ldots, I_n$:
latest transition time for L = MAX {latest transition time for Ij, $j = 1, 2, \ldots, n$} + delay of the gate;

(iii) if Line L is a fanout branch:

latest transition time for L = latest transition time for the fanout stem.

Earliest transition time at a line is the minimum delay on any path from any primary input Line L is a primary input if

(i) line L is a primary input or pseudoprimary input:
earliest transition time for L = 1 unit delay;

(ii) L is an output of a gate G having n inputs $I_1, I_2, I_3, \ldots, I_n$.
Earliest transition time for L = MIN {earliest transition time for Ij, $j = 1, 2, \ldots, n$} + delay of the gate;

(iii) line L is a fanout branch;

(iv) earliest transition time for L = earliest transition time for the fanout stem.

The earliest transition time and latest transition time of all lines are calculated. Using the maximum value of the latest transition time, the longest paths in the circuit are identified.

(2) Every primary input or pseudoprimary input or gate outputs in the longest path are the victims. The lines in the longest path form the set of victim lines V-lines (set_S).

(3) A V-line from set_S is selected. The timing window of victim and aggressor lines is calculated. The aggressor line has a timing window of (earliest $a(t)$, latest $a_i(t)$). Similarly the victim line also has a timing window of ((latest v_j) $- \delta$ unit delays, (latest $v_j(t) + \delta$ unit delays)). The value of δ between the a_i and v_j is assumed to be one unit delay.

(4) The timing window of the selected v_j, as well as the earliest and the latest transition times of other gate outputs are compared. If the timing window (earliest $a_i(t)$, latest $a_i(t)$) of a_i overlaps with the timing window ((latest $v_j(t) - \delta$ unit delay), (latest $v_j + \delta$ unit delay)) of v_j, then the crosstalk-induced transition fault caused by the selected $c(a_i, v_j)$ is added to the target crosstalk fault list.

The earliest and latest transition times for each line are shown for c17 circuit in Figure 1. From the latest transition time the longest paths for c17 circuit are [3, 11, 16, 22],

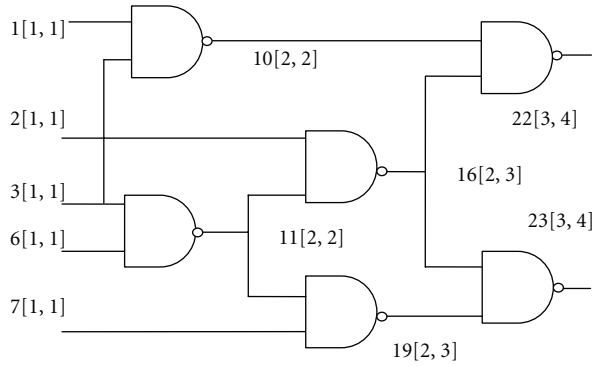

Figure 1: Example circuit c17.

[6, 11, 16, 22], [3, 11, 16, 23], [3, 11, 19, 23], [6, 11, 16, 23] and [6, 11, 19, 23]. Every primary input or pseudoprimary input or gate outputs in the longest path are the victims. The victims in the longest path are [3, 6, 11, 16, 19, 22, 23]. Select a victim 3. The calculated timing window of the victim line is $([1-1],\ [1+1]) = (0,\ 2)$. Check the (3, 1) pair. The timing window of 1(1, 1) overlaps with the timing window of 3(0, 2). Hence (3, 1) forms a target crosstalk fault. Considering the pair (3, 22). The timing window of 3(0, 2) does not overlap with timing window of 23(3, 4). Hence, (3, 22) forms a false crosstalk fault and hence should not be included in the target fault list. The circuit c17 has a total number of target crosstalk faults as 42. The input fault list for the test generator is the reduced target crosstalk fault list.

4. Signal Values Used in Test Generation

The fault model used is logic gate level model. All the gates are assumed to have zero delay. The test generator uses a seven valued logic as shown in Table 1. S0 and S1 are signal values which represent 0 in the first and second test patterns and logic 1 represents 1 in both test patterns, respectively. T0 represents 1 in the first pattern and 0 in the second pattern. T1 represents 0 in the first pattern and 1 in the second pattern. P0 represents glitch on a 0 signal. P1 is glitch on a 1 signal [11].

The operation of the AND gate and OR gate using the signal values mentioned above is shown in the Table 2. Other operations for other gates can be handled in the same way.

5. Test Generation Algorithm

In generating tests for crosstalk delay faults, the following objectives are to be satisfied.

Objective 1: a rising transition or falling transition in the victim line.

Objective 2: a transition in the aggressor lines opposite to that of the victim line.

Objective 3: setting the off-path inputs along the target path to suitable logic values in order to propagate the transition in the victim line to the primary output. The target path is the path with the longest path delay from current site to the primary output [5].

Table 1: 7 valued logic.

Signal	Description
S0	Steady 0
S1	Steady 1
T0	Falling transition
T1	Rising transition
P0	Positive glitch
P1	Negative glitch
X	Unknown value

Table 2: Truth table for (a) two input AND gate and (b) two input OR gate.

(a)

AND	S0	S1	T0	T1	P0	P1	X
S0	S0	S0	S0	S0	S0	S0	S0
S1	S0	S1	T0	T1	P0	P1	X
T0	S0	T0	T0	P0	P0	T0	X
T1	S0	T1	P0	T1	P0	T1	X
P0	S0	P0	P0	P0	P0	P0	X
P1	S0	P1	T0	T1	P0	P1	X
X	S0	X	X	X	X	X	X

(b)

OR	S0	S1	T0	T1	P0	P1	X
S0	S0	S1	T0	T1	P0	P1	X
S1	S1	S1	S1	S1	S1	S1	S1
T0	T0	S1	T0	P1	T0	P1	X
T1	T1	S1	P1	T1	T1	P1	X
P0	P0	S1	T0	T1	P0	P1	X
P1	P1	S1	P1	P1	P1	P1	X
X	X	S1	X	X	X	X	X

In the first step, the objectives are checked for contradiction since objectives have local conflict with other objectives and they are the fanout branches of the same stem.

If any contradiction then the aggressors are removed. If all aggressors are removed then test is not possible. Then the final objectives are assigned to the lines and the remaining lines are set to a value X.

The objectives are propagated and again checked for contradiction. Final list of objectives are obtained and a modified version of FAN algorithm is run as the core to obtain a two vector test for a crosstalk delay fault.

The multiple back-trace procedure which is the most important aspect of the algorithm is applied to back trace from primary outputs toward primary inputs in a breadth first manner to satisfy the multiple primary objectives. The highest level objective generates the previous level objectives and so on. During the back-trace operation, if we create a new objective on an input of a gate, and this input happens to be a fanout branch, then we check first whether or not it has been tried by previous objectives. If so intersect the already assigned value on the stem with the required logic value on the fanout branches. If there is no conflict, the

objective is moved to the primary stack. If conflict occurs in the intersection then the fanout branch with the required logic value is pushed to the secondary stack. After all the objectives in the primary stack are tried then the objectives in the secondary stack are tried and hence all possible choices are considered.

In this work, two stacks are used for test generation which has greater advantage over the single stack, and it reduces the number of backtracks. Further since decision is made only at the fanout points rather than at primary inputs back traces are further reduced. The primary inputs are assigned only the signal values S0, S1, T0 and T1. All gates are assumed to have zero delay.

The pseudocode for the test generation algorithm is shown in Algorithm 1 with the multiple back-trace procedure.

The proposed ATPG algorithm for crosstalk delay fault is explained by taking ISCAS'85 c17 benchmark circuit shown in Figure 1.

The following sample faults are taken from the target fault list for c17 for explanation.
(16, 10), (16, 11), (16, 19), (16, 22), (16, 23).

Step 1. Here 16 is victim and 10, 11, 19, 22, 23 are aggressors. The available paths for propagation of victim to the primary output are (10–22), (19–23).

Step 2. Victim 16 is assigned a rising transition, and the aggressors are assigned falling transition. If the transition at the gate input along the path has a to-noncontrolling transition value any of the three values are allowed at the side inputs: a static noncontrolling and a to-noncontrolling transition and hazardous noncontrolling transition. If the transition at the gate input along the path has a to-controlling transition value any of the three values are allowed at the side inputs: a static controlling and a to-controlling transition and hazardous controlling transition.

Step 3. The path (10–22) is taken for propagation. After checking for contradictions, the final objectives obtained were (16, T1), (10, S1, T1), (11, T0), (19, T0), (22, T0), (23, T0), where (10, S1, T1) is off path objective.

Step 4. Now the current objectives are taken and processed.

Step 5. When (16, T1) is back traced, (11, T0, S1) is the next level objective. This being a fanout point will be checked for contradiction. Since there is no contradiction it will be moved to primary stack. Now when (19, T0) is back traced, (11, T1, S1) is next level objective which results in contradiction, and hence 11 with all values, namely, S1, T0, T1 will be moved to secondary stack. Now when secondary stack is processed, (11, S1, T0, T1) will be processed. The values S1, T0, and T1 are assigned one by one and propagated to check if all objectives are satisfied. (11, S1) will satisfy both objectives (16, T1) and (19, T0).

Step 6. Finally, after backtracking the primary objectives assigned are 1—>S0, 2—>T0, 3—>S0, 6—>S0, 7—>T1. Implications are done and propagated, and delay is observed at the primary output 23. After propagating the faults finally detected were (16, 19), (16, 22), (16, 23).

The next path is taken into consideration and test vectors are arrived for the remaining faults.

6. Experimental Results

The test generation algorithm for crosstalk delay faults was run on ISCAS'85 combinational circuits and several enhanced scan version of ISCAS'89 sequential circuits. Table 3 gives the characteristics of ISCAS'85 combinational circuits and the scan versions of ISCAS'89 sequential circuits. After the circuit name, the number of inputs, number of outputs, number of gates, number of paths, number of critical paths, number of victims, and total number of target faults are given [17]. The entire circuit paths are analyzed using the tree data structure. The total number of paths in the circuit is calculated using depth first search algorithm which employs recursive search procedure. Critical paths are paths whose delay is longer than a given percentage of the longest propagation delay in the circuit. The selection of critical paths, number of victims in the critical path, and the total number of target faults are done using static timing analysis.

The test generation algorithm using modified FAN was implemented in 3000 lines of C language under the LINUX environment. Results were compared with results of modified PODEM based ATPG which was implemented in 2000 lines of C language.

Table 4 gives the percentage increase in fault coverage and reduction in average number of back traces for FAN based test generation compared to PODEM based ATPG. Average number of back traces is given by total number of back traces divided by total number of paths. The bold number represents the maximum number of faults detected and minimum number of back traces obtained for each circuit, respectively. With a nominal increase in fault coverage (**10.4%**) the algorithm achieved a high degree of reduction in average number of back traces (**29.66%**) due to checking of contradictions at the fanout point instead of back tracing to primary inputs. Hence the CPU execution time is greatly reduced.

Table 5 gives the comparison of number of transitions for PODEM- and FAN-based ATPG. For 12 of the 15 benchmark circuits FAN gave less number of transitions. FAN reduces the number of transitions by **28.93%** compared to PODEM. Hence the switching activity in the circuit is reduced thus reducing the power dissipation of VLSI circuits. Number of transitions is reduced because during back tracing it was observed in the results that the values assigned to primary inputs by FAN has more number of static 0s and static 1s compared to PODEM. This is because back tracing in FAN results in more number of unassigned primary inputs with value X. The algorithm assigns static values to unassigned inputs.

Figure 2 gives the comparison between the execution time of FAN- and PODEM-based ATPG. For large circuits

```
Testgen (Faultlist)
begin
   Set primary objectives ( );/*find all the primary
   objectives to be justified*/
   Set all other lines to X
   if (there is conflict between two or more objectives on the fan-out branches of
   the same stem) then
         begin
               Drop the aggressor objectives;
               If (all aggressors objectives dropped)
                  begin
                     then test not possible;
                     Exit
                  End
         end
Imply ( ) /* with respect to the victim and remaining aggressors and other inputs*/
Propagate ( ) /* Propagate the victim along the chosen target path*/
if (there is conflict between the objectives) then
         begin
            Drop the aggressor objectives;
            If (all aggressor objectives dropped) then
               begin
                  test not possible
                  Exit( )
               end
         end
Multiple back trace ( )
Unassigned primary inputs are assigned
Imply ()/* with respect to primary inputs*/
Propagate ( );/* Propagate the delay in the victim along the target path*/
if (delay effect at PO)
                  test is found;
else
         begin
                  no test possible
                  Exit( )
         end
end
  Multiple back trace ( )
  begin
    repeat
      begin
         remove one entry from the current objective;
         if (objective follows a fan-out point) then
            begin
               if (there is a contradiction with a previously assigned value)
                  begin
                     move the objective to a secondary stack
                       else
                       move the objective to a primary stack
                  end
               else
                  begin
                     Continue tracing
                     Select an input of gate with value x
                     add to the current objective
                     return multiple backtrace
                  end
            end
      end
      until current objectives ≠ null
```

Algorithm 1: Continued.

```
if (primary stack objective ≠ null) then
  begin
    remove the highest level objective
    assign most requested value;
    add to the current objective
    return multiple back trace ( )
  end
if (secondary stack objective ≠ null)
  begin
    remove the highest level objective
     repeat
      begin
        assign value to the fan-out point
        Imply ( )
        if (success)
          begin
            add to the current objective
            return multiple back trace ( )
          end
        else
          assign next value
      end
     until all values assigned
   if (assignment not possible)
     begin
       Drop the aggressor objectives;
         If (all aggressor objectives dropped) then
           begin
             test not possible
             Exit( )
           end
     end
  end
end
```

Algorithm 1: Pseudocode for the test generation algorithm.

Table 3: The ISCAS'85 and scan version of ISCAS'89 circuit characteristics.

Circuit	Number of inputs	Number of outputs	Number of Gates	Number of paths	Number of critical paths	Number of victims	Total target faults
c17	5	2	6	11	6	7	42
c432	36	7	160	83926	2199	103	9327
c880	60	26	383	8642	92	70	9279
c499	41	32	202	9440	14	207	21879
c1908	33	25	880	729057	32	93	25916
s27	7	4	10	28	6	10	74
s208	19	10	96	145	2	18	743
s208.1	18	9	104	142	1	12	558
s298	17	20	119	231	1	10	537
s526	24	27	193	410	1	10	891
s386	13	13	159	207	10	49	4195
s510	25	13	211	369	1	13	1098
s420.1	34	17	218	474	1	14	1276
s1196	32	32	529	3097	9	55	10630
s1238	32	32	508	3558	30	45	5822
s13207.1	700	790	7951	1345319	565	188	241743
s15850.1	611	684	9772	164738035	8238	341	240814

Table 4: Comparison of results between FAN and PODEM.

Circuit name	Number of fanout points	% Of average aggressors activated	Average number of back-traces		% Decrease in average number of back traces	CPU time (sec)		Fault coverage (%)	
			FAN	PODEM		FAN	PODEM	FAN	PODEM
c17	3	73	**5.15**	8.461	39.13	**0.000**	0.000	**92**	90
c432	89	94	**132.054**	216.518	39.01	**338.05**	389.2	**80**	79
c499	59	98	**163.789**	233.418	29.83	**816.35**	3040.1	**71**	46
c880	125	97	**314.460**	436.56	27.97	**1268.5**	3145.9	**80**	30
c1908	384	80	**646.377**	842.28	23.21	**3762.3**	5786.5	**15**	7
s27	4	81	**9.593**	12.218	21.48	**0.000**	0.000	**83**	58
s208	32	91	**89.029**	101.208	12.03	**1.400**	1.550	**68**	42
s208.1	28	86	**89.55**	103.93	13.83	**0.65**	0.900	**10**	43
s298	34	94	**76.104**	161.886	52.99	**3.733**	5.500	**85**	59
s526	54	95	**132.353**	301.284	56.07	**13.21**	19.433	**86**	84
s386	26	80	**102.88**	206.06	50.07	**15.26**	21.8	62	**77**
s510	73	88	**156.615**	218.48	28.32	**13.15**	23.05	**65**	47
s420.1	58	93	**175.67**	189.69	7.39	**9.3**	13.58	**22**	12
s1196	155	85	**508.99**	558.26	8.83	**1404**	2862.4	**62**	52
s1238	165	91	**424.97**	658.28	35.44	**704.5**	710.15	**87**	86
Average					**29.66**	**554**	1001.3	**64.53**	54.13

Table 5: Comparison of number of transitions between FAN and PODEM.

Circuit name	Number of transitions		% Reduction in number of transitions over
	FAN	PODEM	PODEM
c17	**94**	117	**19.66**
c432	**199585**	345413	42.21
c499	380926	**374882**	−1.58
c880	130217	**112465**	−13.63
c1908	**430634**	458133	6.00
s27	338	**285**	−15.68
s208	**2131**	3317	35.75
s208.1	**930**	1837	49.37
s298	**6388**	8860	27.90
s526	**8036**	13100	38.65
s386	**15385**	26760	42.50
s510	**4747**	10175	53.34
s420.1	**3435**	8036	57.25
s1196	**107765**	222839	51.63
s1238	**69164**	116570	40.66
	Average		**28.93**

such as c880 and c499, CPU time is reduced by 60% for FAN based ATPG.

Table 6 presents the comparison of the proposed ATPG with the results of [6, 12, 13] for crosstalk delay faults in combinational circuits. In the proposed ATPG, all the target faults are taken into consideration for most of the benchmark circuits except for c1355, c5315, c7552, s13207.1, and s15850.1 where the number of target faults is limited to 10000. In [6] the number of victim/aggressors pairs taken into consideration is only 100 but the CPU time is much higher. For c432, the proposed test generator takes the reduced list of faults as 9327, and CPU time taken was only 338 sec with a better fault coverage. In [12] again 100-single aggressors/victims were generated randomly and the fault coverage calculated. The fault coverage achieved was very low, and CPU time also very high. In [13] the number of target paths is limited to 1000 and number of back traces is limited to 10000 and hence CPU time is less. Except for c1355, c5315, c7552, s13207.1, and s15850.1, the proposed approach takes all the target faults into account and does not limit the back trace and has a better fault coverage for most of the benchmarks. Table 7 represents the comparison of the proposed ATPG with the results of [8, 9, 18] for crosstalk delay faults in sequential circuits.

Table 6: Comparison with previous works.

Circuit name	Previous work [6]		Previous work [12]		Previous work [13]		Proposed ATPG	
	% Fault coverage	Time(s)	% Fault coverage	Time(s)	% Fault coverage	Time(s)	% Fault coverage	Time(s)
c432	35	1019	15	4 hr 42 min	—	—	**80**	338.05
c880	28	1553	38	2 hr 27 min	54.26	0.1	**80**	1268.5
c1355	16	3173	**36**	3 hr 48 min	—	—	33	1865.8
c1908	**33**	2562	8	10 hr 40 min	24.34	360	15	3762.3
c5315	31	7030	1	24 hr 10 min	49	7.85	**57**	7151.2
c7552	14	8424	2	28 hr 26 min	52	1326.6	**68**	9867.03

Table 7: Comparison with previous works.

Circuit name	Previous work [8]		Previous work [9]		Previous work [18]		Proposed ATPG	
	% Fault coverage	Time(s)	% Fault coverage	Time(s)	% Fault coverage	Time(s)	% Fault coverage	Time(s)
s208	31.50	0.18	55.48	**0.2**	—	—	**69**	1.233
s298	42.66	0.17	—	—	—	—	**84**	3.317
s386	34.43	0.55	51.89	**0.3**	—	—	**62**	15.26
s526	66.30	**0.19**	55.24	0.2	—	—	**86**	12.033
s420.1	**27.19**	0.50	—	—	—	—	22	9.3
s1196	31.32	0.67	1.03	**0.4**	8.41	19	**62**	1404
s1238	—	—	3.83	**0.2**	3.11	16	**87**	704.5
s13207.1	87.14	14329	25.13	18287	—	—	64	32333.5
s15850.1	83.05	19627	3.15	83371	—	—	53	97136.4

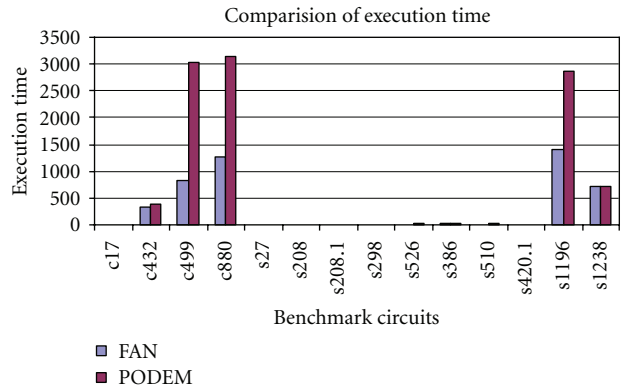

Figure 2: Comparison of CPU Time.

In [8, 9, 18] the calculation of target faults were similar since layout information has not been used and gate delay assumed to be 0. In this paper, gate delay was assumed to be 1. Compared to [8, 9, 18] the proposed approach has higher fault coverage. For larger benchmark circuits s13207.1 and s15850 in [8] only 100 testable randomly selected longest paths are considered for target fault selection. In [18] the waveform sensitization-based algorithm cannot generate tests for the large circuits. Hence compared to previous works the proposed test generation algorithm has produced comparable or greater fault coverage, and CPU time is also nominal.

For sequential circuits even though CPU time is higher the fault coverage increases by an average of 30% for the 6 circuits.

7. Conclusion

A new modified FAN-based ATPG algorithm has been proposed that increases the fault coverage with reduced number of transitions and hence permits safe testing of low power circuits.

In test generation for crosstalk delay fault, the number of objectives is to be a satisfied is larger compared to stuck at faults. Hence the multiple back trace procedure adopted in FAN algorithm concurrently traces more than one path and is more efficient than the back trace along a single path. Moreover, the assignment is done at the fanout points and hence in case of contradiction, fruitless computation is avoided. Hence the modified version of FAN algorithm used for test generation significantly reduced the average number of back traces compared to PODEM. Hence the CPU time, that is, test time reduces significantly by using modified FAN when compared to modified PODEM. Further, the algorithm handles single-victim/-multiple aggressors in a very efficient manner. The number of transitions can be further improved for low power testing by minimising the transition controllability and transition observability cost which is considered as future work.

References

[1] W. Y. Chen, S. K. Gupta, and M. A. Breuer, "Test generation in VLSI circuits for crosstalk noise," in *Proceedings of the IEEE International Test Conference*, pp. 641–650, October 1998.

[2] M. Shakya and K. K. Soundra Pandian, "Low power testing of CMOS circuits: a review," in *Proceedings of the National*

Conference on Signal Processing, Communication and VLSI Design (NCSCV '09), p. VL 106, Coimbatore, India, 2009.

[3] A. Rubio, N. Itazaki, X. Xu, and K. Kinoshita, "An approach to the analysis and detection of crosstalk faults in digital VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 387–395, 1994.

[4] W. Chen, S. K. Gupta, and M. A. Breuer, "Analytic models for crosstalk delay and pulse analysis under non-ideal inputs," in *Proceedings of the IEEE International Test Conference*, pp. 809–818, November 1997.

[5] W. Y. Chen, S. K. Gupta, and M. A. Breuer, "Test generation for crosstalk-induced delay in integrated circuits," in *Proceedings of the International Test Conference (ITC'99)*, pp. 191–200, October 1999.

[6] W. Y. Chen, S. K. Gupta, and M. A. Breuer, "Test generation for crosstalk-induced faults: framework and computational results," *Journal of Electronic Testing*, vol. 18, no. 1, pp. 17–28, 2002.

[7] J.-J. Krstic, J. J. Liou, Y. M. Jiang, and K. T. Cheng, "Delay testing considering crosstalk-induced effects," in *Proceedings of the International Test Conference*, pp. 558–567, November 2001.

[8] H. Li, P. Shen, and X. Li, "Robust test generation for precise crosstalk-induced path delay faults," in *Proceedings of the 24th IEEE VLSI Test Symposium*, pp. 300–305, May 2006.

[9] P. Shen, H. Li, Y.-J. Xu, and X. W. Li, "Non-robust test generation for crosstalk-induced delay faults," in *Proceedings of the 14th Asian Test Symposium (ATS '05)*, pp. 120–125, December 2005.

[10] X. Bai, S. Dey, and A. Krstić, "HyAC: a hybrid structural SAT based ATPG for crosstalk," in *Proceedings International Test Conference*, pp. 112–121, October 2003.

[11] Aniket and R. Arunachalam, "A novel algorithm for testing crosstalk induced delay faults in VLSI circuits," in *Proceedings of the International Conference on VLSI Design*, pp. 479–484, 2005.

[12] A. Sinha, S. K. Gupta, and M. A. Breuer, "A multi-valued algebra for capacitance induced crosstalk delay faults," in *Proceedings of the 17th Asian Test Symposium (ATS '08)*, pp. 89–96, November 2008.

[13] S. Chun, T. Kim, and S. Kang, "ATPG-XP: test generation for maximal crosstalk-induced faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 9, Article ID 5208481, pp. 1401–1413, 2009.

[14] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137–1144, 1983.

[15] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," in *Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25)*, vol. 3, pp. 337–343, 1996.

[16] H. Takahashi, K. J. Keller, K. T. Le, K. K. Saluja, and Y. Takamatsu, "A method for reducing the target fault list of crosstalk faults in synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 252–263, 2005.

[17] S. Jayanthy and M.C. Bhuvaneswari, "Simulation based ATPG for crosstalk delay faults in VLSI circuits using genetic algorithm, ICGST," *AIML Journal*, vol. 9, no. 2, pp. 11–17, 2009.

[18] H. Li, Y. Zhang, and X. Li, "Delay test pattern generation considering crosstalk-induced effects," in *Proceedings of the Asian Test Symposium*, pp. 178–183, November 2003.