Southern Illinois University Carbondale
OpenSIUC

Articles | Department of Electrical and Computer Engineering

2016

ATPG for Delay Defects in Current Mode Threshold Logic Circuits

Ashok Kumar Palaniswamy
Southern Illinois University Carbondale, ashokpa@siu.edu

Spyros Tragoudas
Southern Illinois University Carbondale, spyros@siu.edu

Themistoklis Haniotakis
Southern Illinois University Carbondale, haniotak@siu.edu

Follow this and additional works at: http://opensiuc.lib.siu.edu/ece_articles
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Recommended Citation
Palaniswamy, Ashok Kumar, Tragoudas, Spyros and Haniotakis, Themistoklis. "ATPG for Delay Defects in Current Mode Threshold Logic Circuits." *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS* PP, No. 99 (Jan 2016). doi:10.1109/TCAD.2016.2533863.

This Article is brought to you for free and open access by the Department of Electrical and Computer Engineering at OpenSIUC. It has been accepted for inclusion in Articles by an authorized administrator of OpenSIUC. For more information, please contact opensiuc@lib.siu.edu.

ATPG for Delay Defects in Current Mode Threshold Logic Circuits

Ashok Kumar Palaniswamy, *Member, IEEE,* Spyros Tragoudas, *Senior Member, IEEE,* and Themistoklis Haniotakis, *Member, IEEE*

***Abstract*—An automatic test pattern generation approach to detect delay defects in a circuit consisting of current mode threshold logic gates is introduced. Each generated pattern should excite the maximum propagation delay at the fault site. Manufactured weights may vary, and maximum delay is ensured by applying an appropriately generated set of patterns per fault. Experimental results show the efficiency of the proposed methods.**

***Index Terms*—ATPG, Threshold logic gate, BDD, delay testing.**

I. Introduction

Threshold Logic Gate (TLG) is an emerging alternative to implement Boolean functions. It offers the capability of realizing a complex Boolean function using less number of gates. Although *TLGs* were initially introduced five decades ago [1] they gained more importance in current years due to the recent developments in CMOS-based implementations [2]–[4]. *TLGs* are also implementable with emerging nano-electronic technologies [5], [6].

A Boolean circuit implemented by *TLGs* is called a Threshold Network (TN). Synthesis techniques have been proposed for implementing *TNs* [7]–[11]. The objective of these methods is to minimize the number of *TLGs*.

Recent methods in [2], [3] minimize the delay of Current Mode Threshold Logic (CMTL) gates which are CMOS-based. According to [3], the input patterns of the *CMTL* gate can be categorized into groups based on the delay they exhibit. All patterns in a particular group cause the same delay. *CMTL* gates are clocked. We consider the combinational core of the *TN* which is pipelined, does not have any feedback loops, and has unit combinational depth at each pipeline stage.

Manuscript received July 04, 2015; revised December 19, 2015; accepted January 31, 2016. Date of current version Aaaaaaa xx, 2016. This research has been supported in part by grants NSF IIP 1432026, and NSF IIP 1361847 from the NSF I/UCRC for Embedded Systems at SIUC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This paper was recommended by Associate Editor A. Gattiker.

A. K. Palaniswamy is with Synopsys, Inc., Sunnyvale, CA 94085 USA and also with the Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL 62901 USA (e-mail: ashokpa@siu.edu).

S. Tragoudas, and T. Haniotakis are with the Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL 62901 USA (e-mail: spyros, haniotak@.siu.edu).

Preliminary version of this work was published in Design & Technology of Integrated Systems in Nanoscale Era 2014.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier xx.xxxx/TCAD.2016.xxxxx

The main objective of this work is to implement an Automatic Test Pattern Generation (ATPG) method where each generated pattern excites the maximum possible delay on the *CMTL* gate at the fault site and is more likely to excite an error at the output driven by the faulty *TLG*. Given the pipelined structure of the *TN*, the transition fault (TF) model is ideal for detecting delay defects. This pattern sensitive approach will likely to detect a *TF*, if one exists. The manufactured weights of a *TLG* may vary. It is shown that the maximum delay at the faulty *TLG* is excited even if the test pattern set is generated using the designed weights. To our knowledge, *ATPG* methods for *TNs* only focus on logic defects using the stuck-at (SA) fault model [12], [13].

The main contributions of this work are summarized as follows; First, it is shown how to generate a test set per *TF* to excite the maximum possible delay at the gate and propagate the latched error to an observable point of the pipelined *TN*. An important component of the *ATPG* is to identify the group of patterns that excite the maximum rising or falling transition delay at the selected gate. Second, it is shown how to generate a set of test patterns for each *TF* to ensure the maximum delay in the presence of weight deviations. This is important because physical defects and process variation during the manufacturing of a *TLG* may result into the manufactured weights that differ from the designed (ideal) weight values. Finally, a test set compaction scheme is introduced which focuses on generating a high quality test pattern to detect more than one *TFs* in order to reduce the test data volume and the test application time.

This article is structured as follows. Section II presents preliminaries. Section III presents the *ATPG* method considering the designed weights. Section IV presents a method to generate a test set per *TF* in order to cope with weight deviations. Section V presents the test set compaction technique. Section VI presents experimental results. Section VII concludes.

II. Preliminaries

Threshold Logic (TL) gate is a weight dependent majority gate. It consists of n input variables and a threshold value w_0. Each input variable x_i is associated with a weight value w_i. The output O of *TLG* is defined as [1]:

$$O = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq w_0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Copyright (c) 2015 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org.

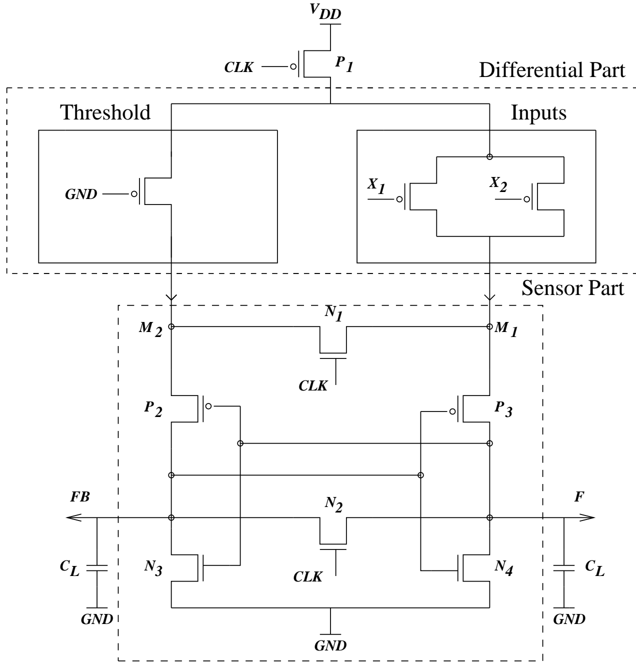


Fig. 1. CMTL gate.

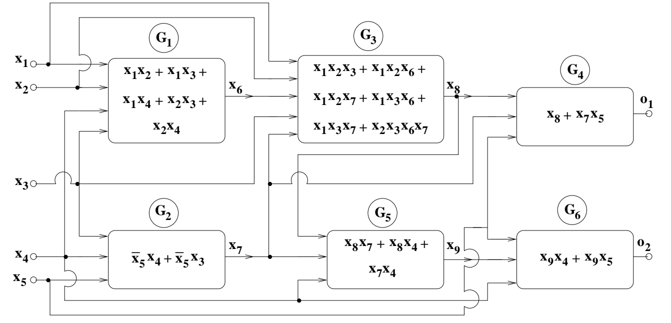


Fig. 2. Combinational circuit.

For simplicity in the notation, let the sum of input variable weights $\sum_{i=1}^{n} w_i$ be denoted as W. Let the active weight summation $\sum_{i=1}^{n} w_i x_i$ of a particular pattern or minterm (fully specified product term) p_l be denoted as W^l.

The *CMTL* gate representation for the two input AND gate with weight values as $(w_1, w_2 : w_0)$ = (2, 2 : 3) is shown in the Figure 1 [3]. The unit weight value corresponds to the minimum gate length of the process technology. The *CMTL* gate consists of the differential and the sensor part. All the transistors in the differential part are connected in parallel. The differential part is further divided into the threshold and the inputs part. The PMOS transistor in the threshold part is always active. The number of active PMOS in the inputs part depends on the applied input pattern. The sensor part has 3 PMOS transistors P_1, P_2, P_3, and 4 NMOS transistors N_1, N_2, N_3, and N_4. The nodes M_1 and M_2 connect the differential part and the sensor part.

The operation of the *CMTL* gate is divided into two phases, the equalization phase (*CLK* is high) and the evaluation phase (*CLK* is low). Due to the design architecture, both output nodes F and FB are at same initial value (≈ 30% of VDD) at the beginning of the equalization phase. Then a small voltage difference is developed across the output nodes which mainly depends on the differential part due to the applied pattern p_l. In the evaluation phase, the sensor part boosts the initial voltage difference to a logic state at the output nodes. The node F rises from 30% VDD to VDD for logic one, and drops from 30% VDD to 0V for logic zero. The node FB changes in the opposite direction with respect to the node F. See [3] for more details.

The propagation delay (d) of the *CMTL* gate for the applied pattern p_l is modeled by [3] is shown in Equation (2). Equation (2) also indicates that the pattern p_l which minimizes $\left|w_0 - W^l\right|$ generates the maximum transition delay at the gate. The transition delay value of the *CMTL* gate decreases linearly with the increase in the $\left|w_0 - W^l\right|$. If the w_0 is equal to the W^l value of the applied pattern p_l, then an unstable value is produced at the output. Hence, the weight assignment for the *CMTL* gate is chosen such that the W^l value for any input pattern p_l is not equal to its w_0 value. Therefore the weight configuration of a *CMTL* gate always needs to satisfy $W^l \neq w_0$ for any pattern p_l.

$$d = c_0 + c_1\, s + \frac{c_2}{s} + 50\, e^{-12} \tag{2}$$

where

$$c_0 = 0.7\, \left(W^2 + w_0^2\right)\, e^{-18} + 20\, e^{-12}$$
$$c_1 = \frac{10\, (W + w_0)\, e^{-6}}{|w_0 - W^l|} + 0.01\, e^{-6}$$
$$c_2 = 0.7\, \left(W^2 + w_0^2\right)\, e^{-18} + 380\, e^{-17}$$
$$s = \sqrt{\frac{c_2}{c_1}}$$

The transition fault model is widely used for testing delay defects at a gate. It requires that a transition is generated at the output of each gate. There are two transition faults associated with each gate: a slow-to-rise (STR) fault, and a slow-to-fall (STF) fault. A *TF* should propagate to an observable point through any path [14].

It is noted that *CMTL* gates are clocked. At the beginning of the clock cycle the output of *CMTL* gate is set to an initial value irrespective of the previous clock cycle value. Hence, the initialization vector of the *TF* model is not required for testing delay defects of the *CMTL* gate. For each *STR* fault at gate G, the test pattern is generated considering that the *ATPG* requires setting logic one at the selected gate and propagating its effect to an observable point. Likewise, a pattern for *STF* is generated.

From the above and the Equation (2), the minterms which evaluate the function to one (also called onset minterms), and, in addition, produce minimum weight summation are responsible for the maximum rising transition delay. The offset minterms (i.e., minterms that set the function to zero) which produce maximum weight summation are responsible for the maximum falling transition delay of the *CMTL* gate.

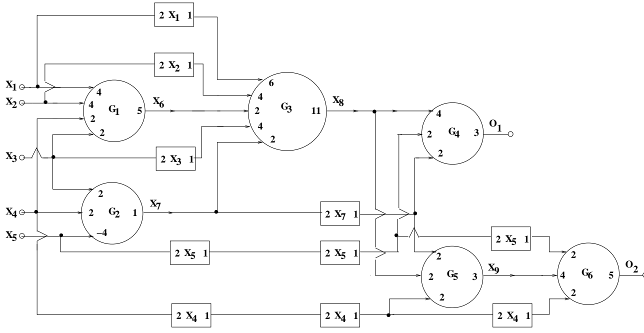


Fig. 3. CMTL threshold network.

The following provide an overview of a combinational *TN* implemented with *CMTL* gates. *CMTL* gates are clocked and circuits that are implemented using *CMTL* gates are pipelined *TNs* without feedback loops. Synchronization *CMTL* buffers are inserted at the pipeline stages so that each *CMTL* gate receives its inputs at the appropriate clock cycle. Methods as in [7] synthesize *TNs* so that the total number of *TLGs* is minimized.

Figure 2 shows a combinational circuit with six complex combinational components whose functionalities are listed explicitly. Each combinational component is a *TLG*, implemented by a *CMTL* gate. Figure 3 shows a four stage pipelined *TN* of *CMTL* gates representing the circuit of Figure 2. The weight assignment for the inputs and the threshold at each *CMTL* gate is also shown. At each pipeline stage, *CMTL* buffers (represented by rectangle) ensure that each *TLG* receives its inputs at the appropriate clock cycle in a synchronized manner.

In this paper, functions will be represented using Binary Decision Diagrams (BDDs). An n input Boolean function f represented in a *BDD* is a directed acyclic graph where the Shannon decomposition is carried out in each node. Each vertex has two outgoing edges. A pointer to each vertex represents a distinct function. One outgoing edge simplifies the function by setting the variable to true and the other outgoing edge by setting it to false. A *BDD* does not contain vertices whose outgoing edges point to isomorphic sub-graphs or to the same node. *BDDs* are canonical forms suitable for representing functions very compactly [15].

III. ATPG BASED ON THE TRANSITION FAULT MODEL

The proposed *ATPG* method is called DTL (delay defect on Threshold Logic gates). This is a function-based test generation approach that uses binary decision diagrams and the conditions listed in Section II to excite maximum *STR* or *STF* delay at the fault site and propagate the transition to an observable point. Each *TLG* has been assigned weight values and is stored in a *BDD*.

Procedure GROUP returns all the minterms of a *TLG* that excite maximum delay at each *TLG* inputs [16]. The set of minterms returned by GROUP is stored as a *BDD* function. Another procedure of DTL, called MAP, rewrites the function returned by GROUP so that the set of minterms of the embedded *TLG* is expressed in terms of the primary inputs of the *TN*.

Algorithm GROUP is a non enumerative *BDD* traversal method that identifies all the fully specified input assignments (minterms) of the given *TLG* function which produce the minimum active weight summation for *STR* or the maximum active weight summation for *STF* [16].

The operation of GROUP is recursive. For the simplicity in the exposition, GROUP is explained when considering a *STR* fault. Similar procedures apply for a *STF* fault. Let f denote the function at a *BDD* node. Let the field M^f be the set of all minterms of f which produce minimum active weight summation value $W^{lf}\left(m_l \in M^f\right)$. For *STF*, M^f will have the set of minterms with W^{lf} denote the maximum active summation value. Let M_t^f denote the simplified function of outgoing edge with the variable set to true. Let M_e^f denote the simplified function of outgoing edge with the variable set to false from the M^f. The *BDD* functions at the two outgoing edges have the minimum active weight sums denoted by W_t^{lf} and W_e^{lf}, respectively.

Algorithm 1: GROUP(F)

Input: Transition fault F at gate G with functionality f
Output: Set of minterms M^f with active weight summation value as $W^{lf}\ \left(m_l \in M^f\right)$

```
1  if f ≠ Constant and Not Visited then
2      M_t^f = GROUP(f_{x_i}) ;
3      M_e^f = GROUP(f_{x̄_i}) ;
4      ADJUSTTHEN(M_t^f, W_t^lf) ;
5      ADJUSTELSE(M_e^f, W_e^lf) ;
6      if W_t^lf ≠ W_e^lf then
7          if F is STR then W^lf = min{W_t^lf, W_e^lf} ;
8          if F is STF then W^lf = max{W_t^lf, W_e^lf} ;
9          if W^lf = W_t^lf then M^f = M_t^f ;
10         if W^lf = W_e^lf then M^f = M_e^f ;
11     else
12         W^lf = W_t^lf ;
13         M^f = M_t^f ∪ M_e^f ;
14     endif
15 endif
16 return M^f ;
```

An input variable which is not present in a *BDD* path is a don't care variable of the path. Algorithm GROUP intends to find the fully specified product terms. Hence in GROUP, initially the $M^f = \{\overline{x}_1\overline{x}_1..\overline{x}_n\}$, $W^{lf} = \infty$ for *STR* and $W^{lf} = -\infty$ for *STF* for each *BDD* node. The don't care variables on the path of each *BDD* node are considered and included in the M^f depends upon their polarity. In order to find the set of input patterns which produce the minimum active weight summation value for *STR* fault, the W_t^{lf} and W_e^{lf} values should be decremented for each don't care variable that has a negative weight value. This is accomplished by including the don't care variable in M_t^f and M_e^f. Similarly, in order to find the set of input patterns which produce the maximum active weight summation value for *STF* fault, the W_t^{lf} and

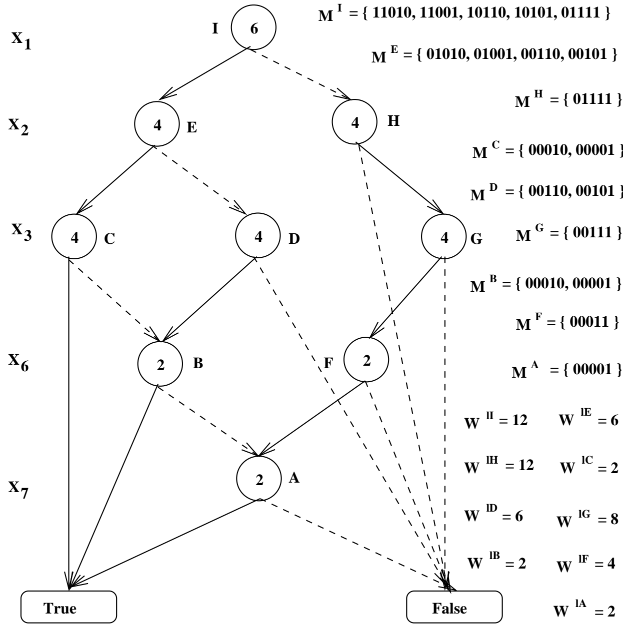


Fig. 4. Example for GROUP illustration.

W_e^{lf} values should be incremented for each don't care variable that has positive weight value.

Procedure ADJUSTTHEN in GROUP performs two updates on variables W_t^{lf} and M_t^f. For a *STR*, first the node variable of the parent node f and its weight value is added to W_t^{lf} and M_t^f, and then it decrements the W_t^{lf} value by including all the don't care variables with negative weight values on the path in the M_t^f. Procedure ADJUSTELSE performs only the second update of the procedure ADJUSTTHEN on W_e^{lf} and M_e^f. The conditions in line 7 and 8 choose the W^{lf} value based upon the fault, i.e., the minimum summation for a *STR* and the maximum summation for a *STF* [16].

GROUP is illustrated for the *STR* fault on gate G_3 at the circuit in Figure 3 with the help of Figure 4 [16]. In this figure, the pointer to the root node (labeled I) represents the functionality of G_3 in terms of its five input variables x_1, x_2, x_3, x_6, x_7. *BDD* node I corresponds to variable x_1, nodes E, H corresponds to variable x_2, nodes C, D, G corresponds to variable x_3, nodes B, F corresponds to variable x_6, and node A correspond to variable x_7. The weight value of the input variables $(x_1, x_2, x_3, x_6, x_7)$ is $(6, 4, 4, 2, 2)$, and the threshold value w_0 is 11. This example shows how to find the onset minterms that produce the minimum active weight summation which excites the maximum rising transition delay at gate G_3. Figure 4 lists for each function f at a *BDD* node the M^f and W^{lf} values [16].

The *BDD* is traversed in reverse topological order [16]. In GROUP, initially $M^f = \{00000\}$ and $W^{lf} = \infty$, for all *BDD* nodes. The node traversal starts at the top pointer node and traverses the *then* path until it reaches the Constant 0 (False) or Constant 1 (True) node. The traversal considers only the paths leading to the *True* node.

In this example, the *then* and *else* children of node A are the *True* and *False* nodes, respectively. This results in $M_t^A = \{00001\}$ and $W_t^{lA} = 2$ by ADJUSTTHEN. This results to $M^A = \{00001\}$ and $W^{lA} = 2$ by choosing the minimum weight summation value.

TABLE I

MAXIMUM DELAY VECTOR SET FOR THE STR FAULT AT GATE G_3

Patterns in Test Set									
Gate Inputs					Circuit Inputs				
x_1	x_2	x_3	x_6	x_7	x_1	x_2	x_3	x_4	x_5
0	1	1	1	1	0	1	1	x	0
1	0	1	0	1	1	0	1	x	1
1	0	1	1	0	1	1	0	0	x
1	1	0	0	1	1	1	0	1	1
1	1	0	1	0					

The *then* child of node B is the *True* node and *else* child is node A. Although x_7 is don't care variable on the *then* path of node B, it is not added by ADJUSTTHEN due its positive weight value. This results in $M_t^B = \{00010\}$ and $W_t^{lB} = 2$, and $M_e^B = \{00001\}$ and $W_e^{lB} = 2$. This results in $M^B = \{00010, 00001\}$ and $W^{lB} = 2$. Similarly, we get $M_t^C = \{00100\}$ and $W_t^{lC} = 4$, and $M_e^C = \{00010, 00001\}$ and $W_e^{lC} = 2$. This results in $M^C = \{00010, 00001\}$ and $W^{lC} = 2$. This way all the *then* path pointer nodes of root node are visited and the M^f and W^{lf} values are updated [16].

Then the *else* path pointer nodes out of the top pointer node are visited. All the pointer nodes are visited and updated in a similar manner. In our example, the top pointer node I results in $M^I = \{11010, 11001, 10110, 10101, 01111\}$ and $W^{lI} = 12$ [16]. The set of patterns responsible for maximum rising transition delay of gate G_3 with respect to the local gate inputs obtained by procedure GROUP is shown on the right hand side of Table I.

In the case of a *STF* fault, the same process is applied except that the maximum active weight summation is considered (line 8). The offset minterms of the given *TLG* function with highest active weight summation result in the maximum propagation delay for a falling transition. Hence the complement of the given gate functionality G_3 is used by GROUP.

In order to represent the functionality of the embedded gate *TLG* in terms of primary inputs, the local input variables (embedded gate inputs) are replaced by primary input variables by function substitutions. In particular, each gate input is replaced by its function. This procedure is referred as MAP. In our example, the five minterms in terms of embedded gate inputs are mapped into seven patterns expressed in terms of primary inputs and shown on the left hand side of Table I.

The input to algorithm DTL is a transition (*STR* or *STF*) fault F at gate G in a *CMTL TN*. The output of the DTL is a random test vector from a collection of patterns, called the vector set S. Set S consists of all patterns that excite the fault with maximum possible delay at gate G and propagate the transition to an observable point of the *TN*.

The overview of algorithm DTL is presented in Algorithm 2. First, procedure PROPAGATION (line 1) generates the set of all pattern vectors P which ensure that the latched error propagates to an observable point. Then procedure GROUP (line 5) generates the set of input patterns set_M_i that produce maximum possible transition delay at the gate G. Subsequently, procedure MAP (line 6) transforms the test vectors

TABLE II
FINAL SET OF PATTERNS FOR THE STR FAULT AT GATE G_3

P					$S = D \cap P$				
x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5
0	1	1	x	0	0	1	1	x	0
1	0	1	x	x	1	0	1	x	1
1	1	x	x	x	1	1	0	0	x
					1	1	0	1	1

in terms of primary inputs (D). Essentially, this procedure generates input patterns that brings each binary pattern in set_M_i at the inputs of gate G. Then the intersection of D and P forms set S (line 7) out of which a test vector is selected.

Algorithm DTL is explained considering the *STR* fault on gate G_3 in the circuit in Figure 3. All patterns that propagate the latched error are kept as a function P and are computed by procedure PROPAGATION. Procedure PROPAGATION propagates the error to an observable point in the *TN*. It is implemented using a boolean difference operation between two functions, the fault free function and the faulty function respectively. For the *STR* at gate G_3 in Figure 3, the propagation vector set is obtained by propagating SA0 at G_3. The obtained input patterns that excite and propagate the error are listed on the left hand side of Table II.

Algorithm 2: DTL(F)

Input: Transition fault F at gate G with functionality f
Output: A test vector

```
1  P = PROPAGATION(G, F) ;
2  S = φ ;
3  i = 1 ;
4  while S = φ and f ≠ φ do
5      set_M_i = GROUP(F) ;
6      D = MAP(set_M_i) ;
7      S = D ∩ P ;
8      if S = φ then
9          f = f \ set_M_i ;
10     endif
11     i++ ;
12 end
13 return  A test vector from S ;
```

The objective of the vector set S is to sensitize the given transition fault (*STR* or *STF*) of the selected gate G with maximum possible delay and propagate the transition to an observable point. For a *STR* fault, for each iteration of procedure GROUP (line 5) produces all the minterms of the function f that excites the maximum delay. The resulting set_M_i is substituted in terms of primary inputs by MAP (line 6). The final sensitization vector set S (intersection of D and P) is shown in the right hand side of Table II.

If $D \cap P$ results in an empty set, then it indicates that no pattern that excites the highest delay at the gate propagates to an observation point. In this case algorithm DTL tries to excite the fault with patterns that excite the next possible highest delay at the gate. This is done by removing the set_M_i from function f (line 9), and then the reduced function is used to find the next highest delay patterns group by GROUP. This is repeated continuously until a non empty S is generated or function f becomes null (line 4). That way, any test vector in S guarantees that a transition (*STR* or *STF*) fault at the gate G is sensitized with the maximum possible delay and the potential error propagate to an observable point.

The systematic removal of the maximum delay patterns group $set_M_i \, (1 \le i \le n)$ maintains the unate property of the given *TL* function f. This is a necessary condition for a function to be a *TL* function. If $set_M_i \, (1 \le i \le n)$ is removed arbitrarily from the function f then f becomes binate [1]. Thus, algorithm GROUP always produces the group of patterns so that all patterns in the group exhibit the same maximum delay. Each pattern in the set set_M_1 excites the highest propagation delay at the fault site.

IV. ATPG TO COPE WITH WEIGHT DEVIATIONS

Let us assume that each weight $w_i \, (0 \le i \le n)$ during manufacturing has an absolute deviation of $\delta_i \, (0 \le i \le n)$, i.e., each manufactured weight value $w_i^{'} \, (0 \le i \le n)$ is in the range $w_i \pm \delta_i$.

This section presents algorithm EDTL, an enhancement of DTL, which generates a set of patterns for each fault such that one of them is guaranteed to excite the maximum delay under any weight deviations.

Let a be the set of onset minterms, and b be the set of offset minterms of the given *TLG* function. Let τ be the maximum allowable shift in the weight summation W^l of any minterm p_l due to the deviation in the manufactured weights which does not affect the *TLG* functionality. From [1], we have that:

$$\tau = \min\{y_1, y_2\} \tag{3}$$

where

$$y_1 = \min_{k \in a}\{W^k\} - w_0$$

$$y_2 = w_0 - \max_{l \in b}\{W^l\}$$

Let us assume for simplicity, that all weight deviations have the same value δ. Then the maximum value of δ for the given weight configuration of the *TLG* is [1]:

$$\delta = \min\{\frac{y_1}{n+1}, \frac{y_2}{n+1}\} \tag{4}$$

It is shown below that although the manufactured weight values may deviate from the designed value, the set of patterns that excite the maximum delay are the patterns that belong to set_M_i that was generated by procedure GROUP when considering the designed weights. (This is the set of patterns expressed in terms of local gate inputs. See line 5 of Algorithm DTL.)

Patterns (minterms) belong into different groups when considering their designed weights. Consider the patterns of two separate groups set_M_i and set_M_j of a *TLG* so that i is less than j. Under weight deviations, the patterns of any such group may further partitioned into sub groups.

Consider the gate G_1 shown in Figure 3. For the given weight configuration $(w_1, w_2, w_3, w_4 : w_0)$ = (4, 4, 2, 2 : 5), we get τ = 1 and δ = 0.2. For the eleven onset minterms of the *TLG* function, we get four delay patterns groups. The

set_M_1 contains minterms p_5, p_6, p_9, and p_{10} with W^5 = 6. The set_M_2 contains minterms p_7, p_{11}, and p_{12} with W^7 = 8. The set_M_3 contains minterms p_{13} and p_{14} with W^{13} = 10. The set_M_4 contains minterm p_{15} with W^{15} = 12.

Assume that the manufactured weight values for $(w_1^{'}, w_2^{'}, w_3^{'}, w_4^{'} : w_0^{'})$ = (4.2, 3.8, 1.8, 1.8 : 5.2). For this new weight configuration, calculating the weight sum for each minterm shows set_M_1 is further divided into two subgroups. The first is called $set_M_{1.1}$ and contains minterms p_5 and p_6 with W^5 = 5.6. The second is called $set_M_{1.2}$ and contains minterms p_9 and p_{10} with W^9 = 6. Similarly, set_M_2 is divided into subgroups. The first is called $set_M_{2.1}$ and contains minterm p_7 with W^7 = 7.4. The second is called $set_M_{2.2}$ and contains minterm p_{11} with W^{11} = 7.8. The third is called $set_M_{2.3}$, and contains minterm p_{12} with W^{12} = 8. None of the minterms in set_M_1 never assigned lower delay patterns group less than set_M_2 for any deviated weight configurations. Similar results are obtained for any two separate groups set_M_i and set_M_j.

Theorem 1: For any n input gate G of an implemented *TN*, no patterns in set_M_i will excite lesser or equal delay at G than a pattern in set_M_j, for any $i < j$.

Proof: Theorem 1 is shown considering only the onset minterms. Similar arguments hold for the offset minterms. For simplicity in explanation, assume that n is even.

From Equation 4, we have that the maximum allowable shift in the weight summation for any minterm due to weight deviation is $\tau = (n + 1) * \delta$. Consider minterms p_x and p_y in any two groups set_M_i and set_M_j so that i is less than j.

Let $g_{ij} = |W^x - W^y|$ for any two minterms p_x and p_y in set_M_i and set_M_j. Due to the weight deviations during manufacturing of a *TLG*, the new $g_{ij}^{'}$ value will be in the range:

$$g_{ij} - \tau \leq g_{ij}^{'} \leq g_{ij} \quad (5)$$

Case 1: Assume that there is no variable that is active in both minterms p_x and p_y. Let n_x and n_y be the number of active variables in p_x and p_y, respectively. In order for the theorem to be violated, the weight summation of minterms in set_M_i must increase, and those in set_M_j must decrease. Therefore W^x is increased by $n_x * \delta$ and W^y is decreased by $n_y * \delta$. However, $n_x + n_y$ is always less than or equal to n. This results in $g_{ij}^{'}$ being no less than $g_{ij} - (n * \delta)$.

Case 2: Assume that there is n_{xy} number of variables that are active in both minterms p_x and p_y. In order for the theorem to be violated, the weight summation of minterms in set_M_i must increase and those in set_M_j must decrease. In the worst case either all the common active variables will have a positive deviation or all will have a negative deviation. In the following, we prove the case assuming that they all have a positive deviation. Similar arguments hold when they all have a negative deviation. We have that W^x is increased by $n_x * \delta$ and W^y is decreased by $(n_y - n_{xy}) * \delta$. However, $n_x + n_y - n_{xy}$ is always less than or equal to n. This results in $g_{ij}^{'}$ being no less than $g_{ij} - (n * \delta)$.

From Equation 5, we have that $g_{ij} - g_{ij}^{'}$ is less than τ. If the weight configuration of a *TLG* satisfies that τ is not greater than $\min\{g_{ij}\}$ then none of the minterms in the group set_M_i will have less delay than the any minterm in the group set_M_j for any weight deviated configurations, when i is less than j. This proves Theorem 1.

Not all the test vectors of S generated by DTL using designed weights may excite the maximum delay for some deviated weight configurations. However, at least one will do. In the above example, the highest delay group (set_M_1) of the designed weights is sub divided into two subgroups ($set_M_{1.1}$, $set_M_{1.2}$) for one of the deviated weight configurations. Only the patterns in the $set_M_{1.1}$ excite highest delay for this deviated weight. Hence all patterns of S must be applied instead of only one. However, only the vectors in S that bring different input assignment at a gate G must be applied. Using the above, DTL is enhanced into algorithm EDTL (Enhanced DTL).

Algorithm 3: EDTL(F)

Input: Transition fault F at gate G with functionality f

Output: Test vector set S

```
1  S = ∅ ;
2  set_M_i = GROUP(F) ;
3  P = PROPAGATION(G, F) ;
4  foreach p_j ∈ set_M_i do
5      D_j = MAP(p_j) ;
6      if D_j ∩ P ≠ ∅ then
7          S_j = A pattern in D_j ∩ P ;
8          S = S_j ∪ S ;
9      endif
10 end
11 return S ;
```

The overview of the EDTL is presented in Algorithm 3. First, GROUP (line 2) generates the set of input patterns set_M_i, where i is the minimum value in algorithm DTL that produces a pattern which excites maximum possible transition delay at gate G. The patterns that excite and propagate the latched error to an observable point are generated by PROPAGATION (line 3). Then procedure MAP (line 5) constructs one or more input patterns D_j for each pattern p_j in set_M_i at the inputs of the gate G. A pattern S_j which bring the distinct maximum delay pattern p_j at the gate G is formed by selecting one of the patterns in the intersection of D_j and P (line 7). That way, a collection of patterns S_j $(j \geq 1)$ is formed, where each S_j will bring distinct p_j of set_M_i at the gate G which sensitizes maximum possible delay and propagate to an observable point in the *TN*. This collection of patterns S_j $(j \geq 1)$ is denoted by S.

The vector set generation by EDTL is illustrated for the *STR* fault on gate G_3 in the circuit in Figure 3. In this example, set_M_i = set_M_1. First, the set of vectors set_M_i and P is determined for the given fault. The patterns in P are listed in right hand side of Table II.

The set of patterns in set_M_1 is shown in column 1 of Table III. There are five input patterns in set_M_1 which excite the maximum delay for rising transition at gate G_3. Patterns D_j and $D_j \cap P$ for each minterm p_j in set_M_1 are shown in columns 2 and 3 of Table III. There are no input patterns

TABLE III
SENSITIZATION VECTOR SET BY EDTL

p_j in set_M_1					D_j = MAP(p_j)					$D_j \cap P$					S_j				
x_1	x_2	x_3	x_6	x_7	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5
0	1	1	1	1	0	1	1	x	0	0	1	1	x	0	0	1	1	0	0
1	0	1	1	0	1	0	1	x	1	1	0	1	x	1	1	0	1	0	1
1	1	0	1	0	1	1	0	0	x	1	1	0	0	x	1	1	0	0	0
					1	1	0	1	1	1	1	0	1	1					
1	0	1	0	1															
1	1	0	0	1															

that justify which bring the patterns "10101" or "11001" of set_M_1 at gate G_3.

Each $S_j, 1 \leq j \leq 3$ consists of one test vector from each $D_j \cap P, 1 \leq j \leq 3$ which brings distinct input assignments that excite the maximum delay at gate G_3 for *STR* under any weight deviations. Patterns S_j are listed in the fourth column of Table III. Observe that the number of patterns generated by EDTL to detect *TF* under weight deviations is reduced to three.

V. TEST SET COMPACTION

This section presents a compact *ATPG* which we call CEDTL (Compact EDTL). For each *STR* or *STF* fault at gate G, algorithm EDTL generates several test functions $D_j \cap P$ (step 6 of Algorithm 3) and then selects a test pattern S_j (step 8 of Algorithm 3). A compact test set is obtained by manipulating the test functions of several gates.

Algorithm 4: CEDTL(TN)
Input: A *TN*
Output: A compact test set for all *TFs*
1 CLUSTERING(TN) ;
2 **foreach** C in *TN* **do**
3 C_i, $1 \leq i \leq 2$ = COMPATIBLE(C) ;
4 **foreach** C_i, $1 \leq i \leq 2$ **do**
5 **return** Compact Test Set = COMPACT(C_i) ;
6 **end**
7 **end**

Algorithm CEDTL is presented in Algorithm 4. Clusters of gates are formed by traversing the *TN* in reserve topological order. The size of each cluster is limited to a predetermined constant value c. The clustering phase helps the scalability of algorithm CEDTL. This procedure is called CLUSTERING.

Consider a *STR* for gate G at some cluster C, and an immediate predecessor gate G at C which is connected to G with an input that has a positive weight. Then CEDTL will generate a compact test set by considering the test functions for a *STR* at G. If the weight of that input is negative then CEDTL will compact by considering the test functions for a *STF* at G. This is due to the unate timing property of threshold logic gates [1]. That way, two sets of test functions are formed for each cluster C: Set C_1 consists of all functions that are compatible with the test functions for a *STR* at the output gate of cluster C, and set C_2 consists of all functions compatible to a *STF* at the output gate of the cluster. This procedure is called COMPATIBLE.

Then a greedy algorithm is applied to the functions in $C_i, 1 \leq i \leq 2$. Any two test functions in C_i are covered by a single function as long as their intersection is non empty. The two functions must target faults at different gates in set C_i since different test functions for the same gate contain disjoint minterms. For any non-empty function intersection, the two functions are substituted by their intersection, they are not considered any further. This process is repeated for the test functions until only empty intersections are encountered among the test functions in each C_i. This greedy algorithm is called COMPACT.

Procedure COMPACT is illustrated with the help of Tables IV and V. Table IV considers a cluster C containing gate G_3 of the *TN* of Figure 3 and its two immediate predecessor gates G_2 and G_1. We consider a *STR* at G_3, i.e., procedure COMPACT operates on the set of functions C_1. Since the input weights of G_3 are positive, the test functions for *STR* at G_2 and G_1 are considered by COMPACT (Line 5).

The second column of Table IV contains the three test functions for gate G_3. According to the notation used in algorithm EDTL, they are labeled as S_1, S_2 and S_3. The third column of Table IV lists the two test functions S_1 and S_2 for gate G_2. Finally, the fourth column lists the two test functions generated by EDTL for gate G_1. Clearly EDTL will return seven patterns for this cluster. For simplicity in the notation, let the test function S_i for the gate G_j be denoted as $S_i\,(G_j)$.

In this example, algorithm COMPACT first considers $S_1\,(G_3)$, and tries to determine whether there is an non-empty intersection among the test functions S_i, $1 \leq i \leq 2$, of the predecessor gate G_2. These functions are examined in increasing order. Therefore it first examines whether $S_1\,(G_3)$ intersects with $S_1\,(G_2)$, and this turns out to be an non-empty test function. At this point, $S_1\,(G_3)$ and $S_1\,(G_2)$ are covered by the intersection of sets $S_1\,(G_3)$ and $S_1\,(G_2)$, and are not considered any further. The test set for the gates in the cluster is already reduced by one pattern.

Next, the test function resulting from the intersection of sets $S_1\,(G_3)$ and $S_1\,(G_2)$ is considered for possible intersections with the two test functions S_1 and S_2 of gate G_1 in column 4. They are considered in increasing order. The first intersection is empty but the second intersection turns out to be non-empty. Let T_1 be the set resulting from the intersection of $S_1\,(G_3)$, $S_1\,(G_2)$, and $S_2\,(G_1)$. Therefore the sets $S_1\,(G_3)$, $S_1\,(G_2)$, and $S_2\,(G_1)$ are not considered any further. The test set for the cluster is reduced by another pattern.

Now the algorithm backtracks to the test functions in the second column of Table IV, and considers $S_2\,(G_3)$. It does

TABLE IV
TEST FUNCTIONS IN SET C_1 OF A CLUSTER WITH GATES G_3, G_2, AND G_1

Test Function Label	CMTL Gates														
	G_3					G_2					G_1				
	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5
S_1	0	1	1	x	0	0	1	1	0	0	x	x	0	1	0
S_2	1	0	1	x	1	1	0	1	0	1	0 1	1 0	1 1	0 0	0 1
S_3	1 1	1 1	0 0	0 1	x 1										

TABLE V
CEDTL FOR THE EXAMPLE IN TABLE IV

Test Functions	Test Patterns				
	x_1	x_2	x_3	x_4	x_5
$T_1 = S_1(G_3) \cap S_1(G_2) \cap S_2(G_1)$	0	1	1	0	0
$T_2 = S_2(G_3) \cap S_2(G_2)$	1	0	1	0	1
$T_3 = S_3(G_3)$	1	1	0	1	1
$T_4 = S_1(G_1)$	1	1	0	1	0

not examine if it will intersect with set T_1 since the latter function covers $S_1(G_3)$ and the result is known to be negative. Thus, it examines whether $S_2(G_3)$ intersects with $S_2(G_2)$. This results in an non-empty set T_2 which covers $S_2(G_3)$ and $S_2(G_2)$. The test set is reduced by another pattern at this point.

Next, the algorithm finds whether T_2 intersects with the only possible test function $S_1(G_1)$. Set $S_1(G_1)$ is disjoint with T_2 and their intersection results in empty set. Now the algorithm backtracks and considers the test function $S_3(G_3)$ in the second column. The algorithm finds whether it intersects with $S_1(G_1)$ which is the only remaining uncovered test function. However, these two test functions ($S_3(G_3)$ and $S_1(G_1)$) are disjoint, and the algorithm terminates.

In this example, CEDTL reduces the test set from seven pattern to four patterns. The right hand side of Table V lists the four test functions, and the left hand side list the test patterns, one pattern per test function.

VI. EXPERIMENTAL RESULTS

Experiments are presented which show that some pattern excite significantly more delay than the others and they must be selected when testing for delay defects. Then a software tool was employed to synthesize pipelined *TNs* for each ISCAS 85 and ITC 99 benchmark. Finally, the section demonstrates the efficiency of the three *ATPG* tools presented in this paper: The basic *ATPG* tool called DTL, the enhanced *ATPG* called EDTL that accommodates weight variation, and the compact *ATPG* method called CEDTL.

All software tools were implemented in the C++ language. The experiments were conducted on a Sun-Blade 100 workstation with a 1 Gigabyte RAM on the ISCAS 85 and ITC 99 Benchmark circuits [17], [18]. As described in Sections II-V, the proposed method uses *BDDs* for each embedded *TLG* in order to identify the maximum delay pattern set for the gate, and a separate *BDD* for the whole circuit in order to

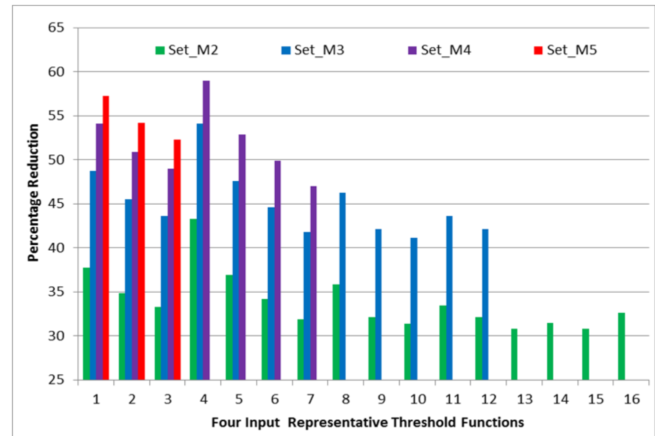


Fig. 5. Percent delay reduction of patterns.

generate the test patterns. Therefore we did not experiment with the multiplier circuit c6288 since traditional *BDDs* cannot easily handle such circuit functionalities. For such circuits, biconditional BDDs have been introduced in [19], and the proposed method can generate patterns by operating on this data structure. However, such implementation details are beyond the scope of this work.

First, an experimental study was conducted to determine the impact of the different input patterns on the delays that they excite at gates. There exist 17 representative threshold logic functions which represent all the threshold logic functions of exactly four inputs [1]. All these functions were considered. The rising transition delay value for each onset minterm of each function was calculated using Equation 2. Many minterms of a function exhibited the same delay. For each function, they were categorized into different groups set_M_i, and all the minterms in the same group set_M_i had the same delay. For each function, up to five groups were formed, i.e., set_M_1 to set_M_5 with the patterns in set_M_1 exhibiting the highest delay.

For a *STR* fault, Figure 5 shows the percentage reduction in the delay value for set_M_i, $i > 2$, in comparison with the delay for set_M_1. The reduction is listed for each possible four input threshold function. Note that only 16 out of 17 functions are listed because when analyzing the results of Figure 5 we observed that for one of them only set_M_1 can be formed. The reduction in the delay value for set_M_2 is at least 43% when compared with the delay of set_M_1, and 34% on average. Similarly, the reduction in the delay for set_M_3

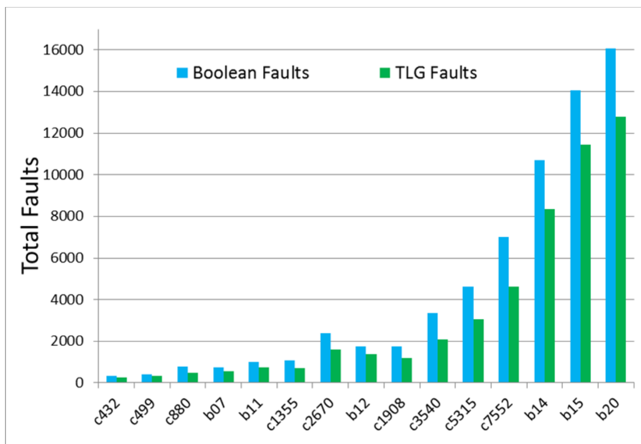


Fig. 6. Total faults of synthesized ITC 99 and ISCAS 85 benchmarks.

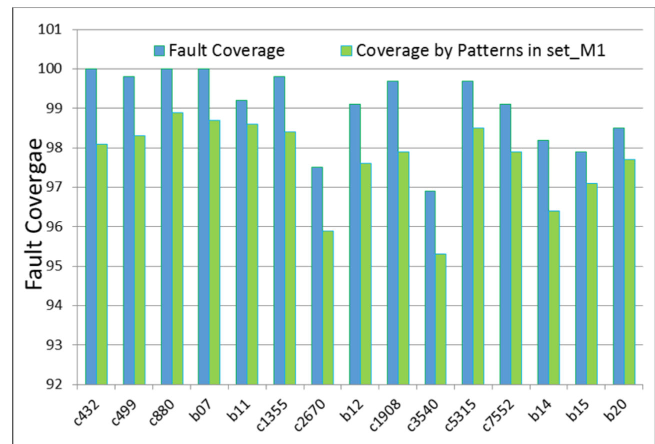


Fig. 7. Fault coverage by DTL.

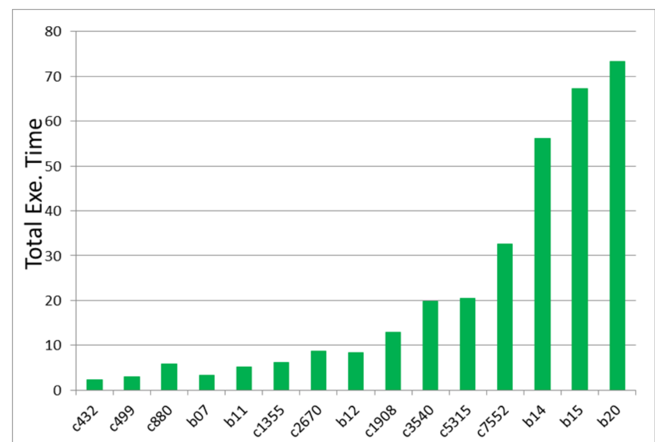


Fig. 8. Time performance of DTL.

is at least 54% when compared with the delay of set_M_1, and 46% on average. Additional experiments have indicated similar behavior for *STF* fault delay values. These results show that it is important to test each transition fault with patterns in set_M_1. Patterns for set_M_2 should only be applied if none of the patterns in set_M_1 propagates to an output, and so on.

Subsequently, a synthesis tool was developed, so that each benchmark is synthesized into a pipelined *TN*. The gates of the input combinational circuit were mapped into *CMTL* gates. It is designed using a modification of [7] since [7] is based on the asummable property of [1] and does not assign weights. Thus, the tool of [7] was modified in order to assign weights of each *TLG*. The *TL* function identification method proposed in [20] was employed to check whether a particular Boolean gate cluster can be implemented as a *TLG*. The fan-in bound of a *TLG* was set to eight.

Figure 6 presents the total number of *TFs* (both *STR* and *STF*) in the pipelined *TN* for each benchmark circuit considered for the proposed *ATPG* methods. The *TFs* of the synchronized buffers are not examined by the *ATPG* because the set of patterns which detects the *TFs* at the *CMTL* gate output will also detect the *TFs* of the synchronized *CMTL* buffer connected to it. Hence, we consider only the *TFs* (both *STR* and *STF*) at the output stem of each *CMTL* gate. Figure 6 also presents total number of *TFs* in the original CMOS Boolean circuits for each benchmark circuit. The reduction in total number of *TFs* in the *TN* is 39% in the best case when compared with the original CMOS Boolean circuit, and 28% on average.

The remainder of the section focuses on the efficiency of the presented *ATPG* algorithms. Figure 7 shows the fault coverage by DTL for each benchmark circuit. It shows the fault coverage is at least 96.9% and that observed in circuit c3540. The fault coverage was 99% on average among all benchmarks. The best fault coverage was 100%. Figure 7 also shows the fault coverage by considering only the patterns in set_M_1. It was at least 95% and that observed in circuit c3540. On average, it was 97% among all benchmarks. The best observed fault coverage was 99%.

Figure 8 shows the time performance of the algorithm DTL. It took approximately 13 seconds to handle all the *TFs* in the benchmark c1908. It was also observed that the maximum execution time never exceeded 74 seconds for any of the benchmark circuit. Thus DTL is a very scalable *ATPG* method.

Figure 9 shows the time performance of algorithm EDTL that considers weight variations due to manufacturing. EDTL took approximately 16 seconds to handle all the *TFs* in benchmark c1908. It was also observed that the maximum execution time never exceeded 92 seconds for any of the benchmark circuit. This is also a very scalable *ATPG*. Note here that the fault coverage of EDTL is same as of DTL since it substitutes a test pattern for DTL by a test set in order to ensure that the fault is detected under any weight variation.

Finally, the experiments focused on the efficiency (test set reduction) and time performance of CEDTL. Let $|S|$ be the total number of patterns needed to detect all *TFs* by EDTL, and $|C|$ is the total number of patterns needed to detect all *TFs* by CEDTL. Then the percentage reduction in test set size by CEDTL is calculated as by $((|S| - |C|) \,/\, |S|) * 100$.

Figure 10 shows the average percent reduction on the number of test vectors that were needed to detect all the *TFs* under any weight deviation in each benchmark circuit. The cluster size bound was set to four in all benchmark circuits. For circuit b11, the reduction was at least 40%. On average

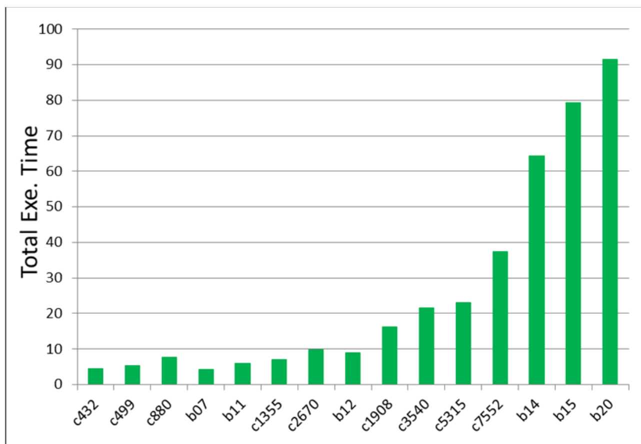


Fig. 9. Time performance of EDTL.

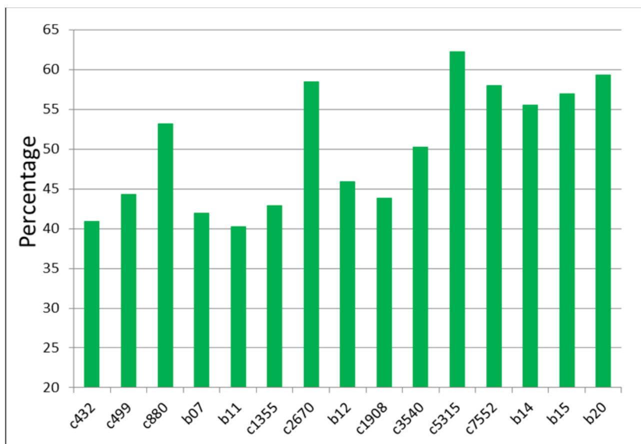


Fig. 10. Test set size reduction by CEDTL.

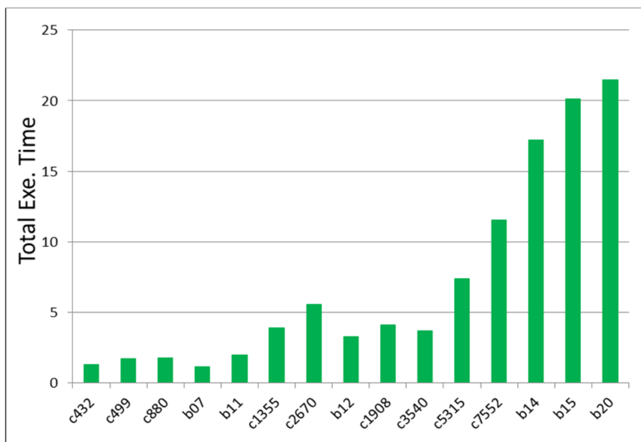


Fig. 11. Time performance of CEDTL.

among all benchmark circuits, the reduction was 52%. In the best case, it was 62%.

Figure 11 shows the time performance of CEDTL (excluding the time taken by EDTL). It took approximately 4 seconds to handle all the *TFs* in the benchmark c1908. It was also observed that the maximum execution time never exceeded 22 seconds for any of the benchmark circuit. This is also a very scalable *ATPG*.

VII. Conclusion

In this article, we presented *ATPG* tools for current mode threshold logic gate circuits which are designed using CMOS technology. They use the transition fault model that can handle small delay defects due to the pipeline nature of the designs. Since different patterns excite different delays at the fault site, *ATPG* tools focus on generating patterns that excite the maximum possible delay for each fault. Three *ATPG* tools have been presented. The basic *ATPG* tool is very scalable and ensures very high fault coverage. A second *ATPG* tool was developed to handle instances where the manufactured weights differ from designed weights due to process variations. A compact *ATPG* has also been presented that reduce the test size for all benchmark circuits by approximately 52%.

References

[1] S. Muroga, *Threshold Logic and Its Applications.* New York: Wiley-Interscience, 1971.

[2] S. Bobba and I. N. Hajj, "Current-mode threshold logic gates," in *Proc. IEEE ICCD*, pp. 235–240, Sep. 2000.

[3] C. B. Dara, T. Haniotakis, and S. Tragoudas, "Delay analysis for an n-input current mode threshold logic gate," in *Proc. IEEE ISVLSI*, pp. 344–349, Aug. 2012.

[4] J. Yang, N. Kulkarni, S. Yu, and S. Vrudhula, "Integration of threshold logic gates with RRAM devices for energy efficient and robust operation," in *Proc. IEEE/ACM NANOARCH*, pp. 39–44, May. 2014.

[5] J. Quintana, M. Avedillo, and J. Nunez, "Design guides for a correct DC operation in RTD-based threshold gates," in *Proc. IEEE Euromicro DSD*, pp. 530–536, 2006.

[6] D. Bol, J. D. Legat, J. M. Quintana, and M. Avedillo, "Monostable-Bistable transition logic elements: Threshold logic vs. Boolean logic comparison," in *Proc. IEEE ICECS*, pp. 1049–1052, Dec. 2006.

[7] L. Pierce and S. Tragoudas, "Nanopipelined threshold network synthesis," *ACM Jour. Emerg. Technol. Comput. Syst.*, vol. 10, no. 2, pp. 17:1–17:17, Mar. 2014.

[8] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. Comput-Aided Des.*, vol. 24, no. 1, pp. 107–118, Jan. 2005.

[9] M. K. Goparaju, A. K. Palaniswamy, and S. Tragoudas, "A fault tolerance aware synthesis methodology for threshold logic gate networks," in *Proc. IEEE DFT*, pp. 176–183, Oct. 2008.

[10] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. Comput-Aided Des.*, vol. 30, no. 5, pp. 665–677, May. 2011.

[11] A. K. Palaniswamy and S. Tragoudas, "Improved threshold logic synthesis using implicant-implicit algorithms," *ACM Jour. Emerg. Technol. Comput. Syst.*, vol. 10, pp. 21:1– 21:32, Apr. 2014.

[12] K. Weidong and E. Banatoski, "Testing for threshold logic circuits based on resonant tunneling diodes," in *Proc. IEEE NANO*, vol. 1, pp. 387–390, Jun. 2006.

[13] P. Gupta, R. Zhang, and N. K. Jha, "Automatic test generation for combinational threshold logic networks," *IEEE Trans. VLSI Syst.*, vol. 16, no. 8, pp. 1035–1045, Jul. 2008.

[14] S. Krstic and K. Cheng, *Delay Fault Testing for VLSI Circuits.* London: Springer, 2012.

[15] R. E. Bryant, "Graph based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[16] A. K. Palaniswamy, S. Tragoudas, and T. Haniotakis, "ATPG for transition faults of pipelined threshold logic circuits," in *Proc. IEEE DTIS*, pp. 1–5, May. 2014.

[17] J. Hayes, *ISCAS Benchmark Circuits.* [Online] Available: http://www.eecs.umich.edu/~jhayes/iscas/

[18] Polito, *ITC Benchmark Circuits.* [Online] Available: http://www.cad.polito.it/downloads/tools/itc99.html

[19] L. Amaru, P. Gaillardon, and G. De Micheli, "A novel canonical bdd for logic synthesis targeting xor-rich circuits," in *Proc. IEEE DATE*, pp. 1014–1017, Mar. 2013.

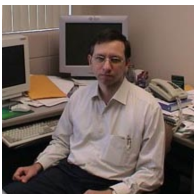
[20] A. K. Palaniswamy and S. Tragoudas, “An efficient heuristic to identify threshold logic functions,” *ACM Jour. Emerg. Technol. Comput. Syst.*, pp. 19:1–19:17, Aug. 2012.

Ashok Kumar Palaniswamy received the B.E. degree in electronics and communication from the Anna University, Chennai, India, in 2006, the M.S. and Ph.D. degrees in electrical and computer engineering from Southern Illinois University, Carbondale, IL, USA, in 2009 and 2014, respectively.

He is currently with Synopsys, Inc., Sunnyvale, CA, USA. His Ph.D. dissertation is in the area of synthesis and testing of circuits consisting of threshold logic gates. His research interests include Synthesis and Verification of digital circuits, VLSI Design and Test Automation, and VLSI Testing.

Spyros Tragoudas (BSc 1986, MSc 1988, PhD 1991) is Professor and Department Chair at the Electrical and Computer Engineering (ECE) Department, Southern Illinois University at Carbondale (SIUC), and the Director of the National Science Foundation (NSF) Industry University Cooperative Research Center (IUCRC) on Embedded Systems at the SIUC site. He has held prior appointments with the faculty of the ECE Department at the University of Arizona, and with the faculty of the Computer Science Dept. at SIUC.

His current research interests are in the areas of VLSI Design and Test Automation and embedded systems. Dr. Tragoudas has published over two hundred papers in journals and peer-reviewed conference proceedings in these areas, and has received three outstanding paper awards for research in VLSI Testing. His research has been funded from federal agencies and industry. He has served and current serving on the editorial board of several journals, the technical program committees of many conferences, was the program chair of the DFTS’09, and the general chair of DFTS’10.



Themistoklis Haniotakis received a B.S. degree in physics and a Ph.D. degree in Informatics from the University of Athens, Greece. His Ph.D. thesis is in the area of Self Checking Circuits. He is a faculty at the Electrical and Computer Engineering Department at Southern Illinois University at Carbondale and has held prior appointment as a faculty in University of Patras, Greece.

His interests include VLSI design, Fault-Tolerant computing, VLSI Testing and Design For Testability, RF IC Design and Test. He has 20 Journal and more than 50 Conference publications. He is a member of IEEE, has receive best paper award (ISQED), has been a reviewer in IEEE journals and conferences and has been a member of various Program Committees.