Sensitization Criterion for Threshold Logic Circuits and its Application

Chen-Kuan Tsai[1], Chun-Yao Wang[1], Ching-Yi Huang[1], Yung-Chih Chen[2]
[1]Dept. of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[2]Dept. of Computer Science and Engineering, Yuan Ze University, Chung-Li, Taiwan

***Abstract*—Threshold logic has been known as an alternative representation of Boolean logic due to its compactness characteristic. Recently, the developments in advanced nanotechnologies have also promised efficient implementations of threshold logic gates. Thus, many synthesis methodologies for threshold logic circuits have been proposed. Since threshold logic has a different mechanism in functional evaluation compared to the traditional Boolean logic, a threshold logic gate can represent a more complex function. As a result, the sensitization criterion in threshold logic circuits is also different. In this work, we propose a sensitization criterion for threshold logic circuits, and show its application to the static timing analysis problem. The experimental results show the accuracy of the proposed criterion.**

I. Introduction

Threshold logic is an alternative form, which possesses the compactness characteristic, to present a Boolean network with a smaller depth and fewer nodes. For example, a Boolean function $f = ab + bcd + e + g$ can be represented by a single threshold logic gate. In the past decades, many different approaches to hardware implementations of combinational/sequential threshold logic circuits had been proposed [23, 25]. However, due to the lack of an *efficient* implementation for threshold logic gates, the developments of design automation methods for threshold logic had slowed down compared to its counterpart in Boolean logic. Recently, with the advanced development in nanotechnologies, many nano-devices have been proposed, such as resonant tunneling diode [3, 37], single-electron transistor [9–11, 17, 31, 41], and quantum cellular automata [39, 43], which provide promising and efficient implementations for threshold logic gates. Meanwhile, efficient CMOS implementations of threshold logic gates have also been available [7]. A more detailed discussion and a comprehensive survey for threshold logic implementations were summarized in literature [4].

Despite the efforts had been made on the implementations of threshold logic gates, only a few commercial solutions had adopted the threshold logic implementations, such as MIPS R2010 [26], SUN Sparc V9 [32], and the Itanium 2 microprocessor [36]. In the comprehensive survey of threshold logic implementations around a decade ago [4], Beiu et al. concluded that the reason why the competitive threshold logic gates are not widely used is due to the lack of synthesis tools. As a result, threshold logic circuit designs require a great amount of manual efforts. Thus, the pervasion of threshold logic depends not only on the efficient implementations, but also on the availability of design automation tools.

Fortunately, in parallel with the advances of threshold logic implementation, the design automation research on threshold logic has been advanced as well. Different synthesis methodologies have been proposed to synthesize multi-level threshold logic networks [2, 18, 20, 21, 29, 46]. In the field of verification and testing, efforts have been made as well [19, 22, 47]. Gupta et al. developed a fault model targeting RTDs technology and proposed a generic automatic test pattern generation (ATPG) framework for testing threshold logic networks [22]. Gowda et al. and Zheng et al. proposed different algorithms performing logic equivalence checking for threshold logic circuits [19, 47].

Sensitization criterion is important to timing analysis, and other applications [15, 16, 24, 28, 45]. For traditional Boolean circuits, many different sensitization criterions have been proposed for computing the longest sensitizable path, or *critical path* of the circuits [5, 8, 13, 15, 16, 30, 33, 34, 40, 42]. However, for threshold logic circuits, since its evaluation mechanism is different from the Boolean circuits and not every threshold logic gate has the controlling value or non-controlling value due to the compactness characteristic of threshold logic[1], its sensitization criterion is different as well.

To the best of our knowledge, this work is the first study targeting the sensitization criterion of threshold logic circuits. We also show its application to the static timing analysis (STA) of threshold logic circuits. The main contributions of this work are two-fold:

1) It is the first work that proposes a sensitization criterion for threshold logic circuits.
2) An STA algorithm for threshold logic circuits is developed and integrated with an existing threshold logic synthesis tool, TELS [46].

II. Background

In this section, we review the background about the threshold logic, and the sensitization criteria in the traditional Boolean logic networks.

A. Threshold logic

A *Linear Threshold logic Gate* (LTG) has n binary inputs, $x_1, x_2, ..., x_n$, and one binary output f. Each input variable x_i is associated with a weight w_i, and every LTG is associated with a threshold value T. A *threshold logic function* is a subset of Boolean functions that can be synthesized as a single LTG [35]. A *threshold logic network* is composed of LTGs, and each LTG can be represented as a $weight - threshold\ vector$ $\langle w_1, w_2, ..., w_n; T\rangle$. For example, Fig. 1(a) shows a Boolean network. Its corresponding threshold logic network can be represented as a multiple-gate network, e.g., Figs. 1(b) or 1(c). In Fig. 1(c), we can use $\langle 5, 2, 3, 1, 1; 5\rangle$ to represent the threshold logic gate G_5.

This work is supported in part by the National Science Council of Taiwan under Grant NSC 102-2221-E-007-140-MY3, NSC 102-2221-E-155-087, NSC 101-2221-E-155-077, NSC 101-2628-E-007-005, and NSC 100-2628-E-007-031-MY3, and by the National Tsing Hua Unversity under Grant NTHU 102N2726E1.

[1]A threshold logic gate can represent a function that is a composition of many Boolean logic gates.

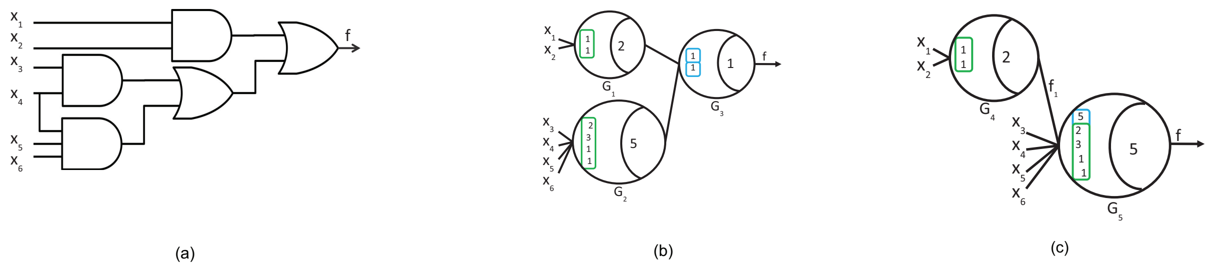

Figure 1: (a) An original Boolean network. (b) A synthesized threshold logic network. (c) Another synthesized threshold logic network.

The evaluation of output f of an LTG is based on the relationship of the weighted summation and threshold value, as formulated in EQ(1). The output f is 1 if the summation of each product $x_i \times w_i$ is greater than or equal to the threshold value T. Otherwise, the output f is 0. For example in Fig. 1(c), G_4 is 1 if and only if $X_1 \times 1 + X_2 \times 1 \geq 2$.

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i w_i \geq T \\ 0, & \text{if } \sum_{i=1}^{n} x_i w_i < T \end{cases} \tag{1}$$

Unateness is an important property of LTGs, because all threshold logic functions are unate Boolean functions [27]. Nonetheless, not every unate Boolean function is a threshold logic function. For example, $f = ab + cd$ is a unate function, but it is not a threshold logic function since it cannot be respesented by a single LTG. If all the weights of an LTG are positive (negative), the Boolean function it represents is *positive (negative) unate*. The characteristics of threshold logic were explored and summarized in [14, 35, 44].

For ease of analysis, we assume that the weights of each LTG are positive in this work. This assumption can be achieved by performing the positive-negative weight transformation procedure [35] when the given LTG has a negative weight.

Next, we introduce some terminologies about LTGs, which will be used in the succeeding discussion.

Definition 1: A *group* of an LTG is composed of either a single input having the weight greater than or equal to the threshold value T, or all the inputs having the weights smaller than the threshold value but the summation of all weights is greater than or equal to the threshold value, referred as *single-input group* and *multiple-input group*, respectively.

Input grouping is the process that separates the inputs and their corresponding weights of an LTG into different groups. An LTG can have one or more groups with respect to the different weights and threshold values. For example in Fig. 1(c), the inputs of G_5 are separated into two groups, f_1 forms a single-input group and $X_3 \sim X_6$ form a multiple-input group. In Figs. 1(b) and 1(c), G_1, G_2, and G_4 only have one multiple-input group; while G_3 and G_5 have two groups.

Definition 2: A *multiple-group* LTG G is a threshold logic gate having multiple groups. If G has only one group, it is referred as a *single-group* LTG.

Definition 3: An LTG is *useless* if and only if it outputs zero for all input combinations [29].

In this work, we assume that the given threshold logic network contains no *useless* LTGs. For instance, $\langle 1; 2 \rangle$ and $\langle 1, 2, 2; 6 \rangle$ are useless LTGs due to the satisfaction of Definition 3. Theorem 1 is used to determine whether an LTG is useless or not.

Theorem 1[29]: Given an LTG, it is useless if and only if it satisfies $\sum_{i=1}^{n} w_i < T$, where n is the number of inputs of the LTG.

Definition 4: An LTG G has a *critical* input x_i if and only if G is useless after removing x_i and the corresponding weight w_i.

Theorem 2 is used to determine whether an input x_i of an LTG is critical or not.

Theorem 2[29]: Consider an LTG, an input x_i is *critical* if and only if it satisfies $\sum_{j=1, j \neq i}^{n} w_j < T$, where n is the number of inputs of the LTG.

For instance in Fig. 1(b), X_4 is a critical input of G_2, because the summation of all the other weights except w_4, $2 + 1 + 1 = 4$, is less than the threshold value, 5.

B. Sensitization Criteria in Boolean Logic

A gate is said to be *sensitizable* if there exists one input assignment that propagates a transition against the previous input assignment from the input to the output of the gate.

Different sensitization criteria and algorithms have been proposed to identify the critical path of a circuit [8, 16, 33, 40]. In general, the sensitization criteria can be classified into two categories: one is the correct sensitization [16, 40], and the other is the exact sensitization [8, 33]. A *correct* sensitization criterion never estimates a smaller gate delay than the actual delay, but could overestimate the delay instead. The accuracy of this sensitization criterion is determined by how close the estimated delay is against the actual delay. On the other hand, an *exact* sensitization criterion can exactly estimate the same delay as the exhaustive timing simulation approach does [8].

Given a path $P = (x_0, G_1, x_1, ..., G_i, x_i, G_{i+1}, ..., G_{n-1}, x_{n-1}, G_n, x_n)$, where x_0 is a primary input (PI), x_n is a primary output (PO), and x_i is a wire connecting two gates G_i and G_{i+1} in the path P. x_i is named as an *on-input* of the path P, and the other inputs of the gates along the path are referred as *side-inputs*. To sensitize a path P, each on-input x_i has to meet the sensitization conditions of the gate. Different sensitization criteria have different sensitization conditions with respect to the gate types. A *controlling value* of a gate G, denoted as $cv(G)$, is a logic value that determines the output value of the gate, independent of the side-inputs' values. For instance, assume G_1 is an AND/NAND gate, $cv(G_1) = 0$. Assume G_2 is an OR/NOR gate, $cv(G_2) = 1$. The complement of a controlling value is called a *non-controlling value*, denoted as

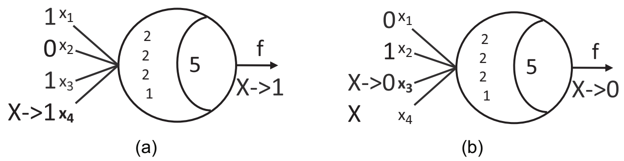

Figure 2: (a) The first condition for output stability. (b) The second condition for output stability.

$ncv(G)$. For example, $ncv(G_1) = 1, ncv(G_2) = 0$. In this work, we use four-valued logic including 0, 1, X (unknown), and $-$ (don't care) to model the signal values.

Next, we introduce the exact sensitization criterion [8], which is the basis of our work. To sensitize a transition of a gate along a path P, each on-input x_i of the gates on P must meet one of the following two conditions:

1) x_i arrives earliest among those inputs of the gate G_{i+1} holding $cv(G_{i+1})$ while some side-inputs of the gate G_{i+1} may be $ncv(G_{i+1})$.
2) x_i arrives latest among all inputs of the gate G_{i+1} and $x_i = ncv(G_{i+1})$ while all the side-inputs of the gate G_{i+1} are also $ncv(G_{i+1})$.

III. Proposed Sensitization Criterion

In this section, we investigate and classify the types of LTGs, and propose the corresponding sensitization conditions for them. Some types of LTGs have the same functionalities as the primitive gates in Boolean logic, such as AND/NAND and OR/NOR gates. Therefore, their sensitization conditions are the same as the ones in Boolean logic. However, other types of LTGs, e.g., those representing complex functions, do not have the concepts of the controlling or the non-controlling values. Hence, in the following paragraphs, we will discuss the proposed sensitization conditions for them.

Before we present the sensitization conditions for different types of LTGs, we discuss the conditions that make the outputs stable using the example in Fig. 2 according to the output evaluation mechanism of LTGs. There are two different conditions leading to a stable output of LTGs under the floating mode operation [8, 15]. The first stable condition is that once the summation of weights in the stabilized-at-1[2] inputs is greater than or equal to the threshold value, the LTG will be stable as 1. The second stable condition is that once the summation of the weights in the stabilized-at-1 inputs and the other unstabilized inputs is less than the threshold value, the LTG will be stable as 0. We use the example in Fig. 2 to explain these conditions under the assumption that the arrival order of input variables is $x_1 > x_2 > x_3 > x_4$. Consider x_4 in Fig. 2(a), both inputs x_1 and x_3 earlier arrive and are stabilized-at-1. Once x_4 is stabilized-at-1, the LTG will be stable as 1 because the summation of weights in the stabilized inputs is equal to the threshold value. On the other hand, consider x_3 in Fig. 2(b), x_2 earlier arrives and is stabilized-at-1. Once x_3 is stabilized-at-0, the LTG will be stable as 0. This is because the summation of weights in the stabilized-at-1 and the remaining inputs, i.e., $w_2 + w_4$, is less than the threshold value.

Next, we introduce a terminology which is used to describe the situation that the output of an LTG is stable caused by an input.

[2]Stabilized-at-0 and stabilized-at-1 are defined as the final state, either 0 or 1, of inputs during the signal evaluation process of LTGs.

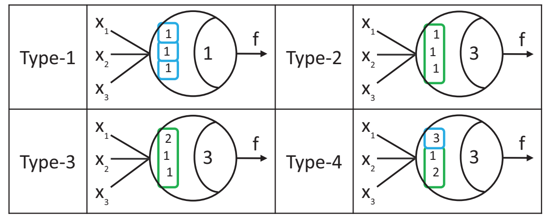

Figure 3: Using 3-input LTGs to represent the four different types of LTGs.

Definition 5: When x_i is stabilized, the output state of a threshold logic gate G becomes stable. Then, this input x_i is named as a *dominant input* of G.

Theorem 3 is used to determine whether an input x_i of an LTG is dominant or not.

Theorem 3: A dominant input x_i exists if and only if an LTG satisfies either EQ(2)

$$(\sum_{j=1}^{i-1} x_j \times w_j) + w_i \geq T \tag{2}$$

where x_i is stabilized-at-1, x_j is the input arrives not later than x_i, and $\sum_{j=1}^{i-1} x_j \times w_j < T$, or EQ(3)

$$\sum_{j=1}^{i-1} x_j \times w_j + \sum_{k=i+1}^{n} w_k < T \tag{3}$$

where x_i is stabilized-at-0, x_j is the input arrives not later than x_i, x_k is the later input than x_i, and n is the number of inputs.

Moreover, if there are two or more inputs that arrive at the same time and are stabilized-at-0/1, under the floating mode operation, these inputs can be all considered as the dominant inputs of the LTG whenever they satisfy the stable conditions. For example in Fig. 2(b), assume that x_3 and x_4 are simultaneously stabilized-at-0, they cause the weight summation less than the threshold value 5. Since both x_3 and x_4 satisfy EQ(3), they are both considered as the dominant inputs of the LTG.

In this work, we classify the LTGs into four types as shown in Fig. 3 according to the sensitization conditions investigated.

Type-1: Multiple-group LTG, given all of its groups are single-input groups: Since every input itself forms a group, it means that when one of the groups is set to 1, the output will be 1. This type of LTG is functionally equivalent to an OR gate. As a result, the sensitization conditions in Boolean logic can be applied to this type of LTG as well.

Type-2: Single multiple-input group LTG, given all of its inputs are critical: Since all of its inputs are critical, it means that when one of the inputs is set to 0, the output will be 0. This type of LTG is functionally equivalent to an AND gate. As a result, the sensitization conditions in Boolean logic can be applied to this type of LTG as well.

Type-3: Single multiple-input group LTG, given one or more of its inputs is not critical: This type of LTGs is similar to Type-2 LTG except that one or more of its inputs are not critical inputs. Since this type of LTG represents a complex function, it has neither a controlling value nor a non-controlling value. However, it still can make the output stable under certain conditions. The proposed sensitization condition for this type of LTG is as follows.

3) x_i is a dominant input of the gate G_{i+1}.

Type-4: Multiple-group LTG, given one of its groups is a multiple-input group: This type of LTG is composed of one or more single-input groups and one multiple-input group, and also represents a complex function. The newly proposed sensitization condition for Type-3 LTGs can also be applied on this type of LTG.

As a result, the proposed new sensitization condition can be integrated with the first two traditional sensitization conditions mentioned in Section II-B to form our sensitization criterion. Based on the observation for the output stability of threshold logic and the integrated sensitization conditions, the proposed sensitization criterion is also *exact*.

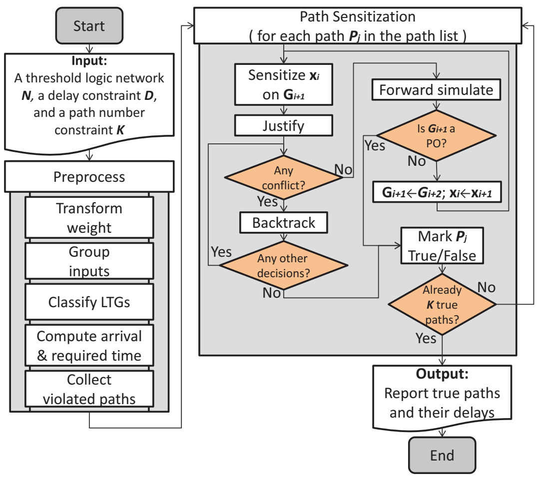

Figure 4: The overall flow of the proposed STA algorithm.

IV. Application to Static Timing Analysis

In this section, we present an application of STA for threshold logic circuits using the proposed sensitization criterion as shown in Fig. 4. Given a threshold logic network N, a delay constraint D, and a path number constraint K, the algorithm will report at most K critical paths whose path delays are greater than D. The value of D is determined by designers and used to set the delay constraint that a path would violate. The value of K is used to determine the desired number of critical paths.

The delay model used in this work is a normalized delay model of threshold logic, which was concluded by the simulation results using HSPICE [6, 38]. The simulation results show that the gate delay is proportional to the number of fanin when the fanin number is smaller than 20. This delay model is generally applicable to many threshold networks, since the fanin numbers of general threshold networks are not larger than 20. For simplicity, the wire delay in this work is assumed to be zero[3].

The proposed algorithm consists of two major parts: preprocess and path sensitization, which will be discussed in detail in the following subsections.

[3]The proposed algorithm is also applicable to the threshold network with a non-zero wire delay model.

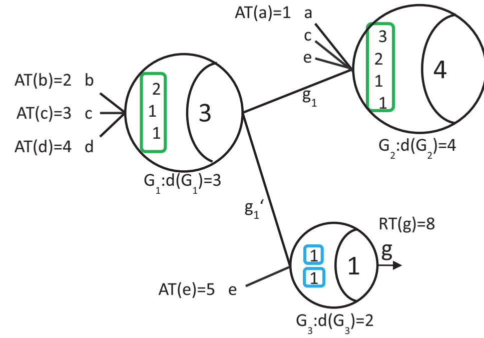

Figure 5: An illustration for the computation of the arrival and required times.

A. Preprocess

During the preprocess, we first transform the given threshold logic network. The transformation procedure includes the positive-negative weight transformation, inputs grouping, and LTGs classification. Next, we compute each gate's minimal and maximal arrival times in a depth-first search (DFS) manner from PIs to POs based on the delay values. Then, we compute the required time of each gate from POs to PIs. Finally, we collect a list of violated paths according to the path delay and the delay constraint D.

We use Fig. 5 to explain the computation of minimal, maximal arrival times, and required time under the assumption that the delay constraint $D = 8$ and $AT(a) \sim AT(e) = 1 \sim 5$, respectively. Given a path $P = (x_0, G_1, x_1, ..., G_i, x_i, G_{i+1}, ..., G_{n-1}, x_{n-1}, G_n, x_n)$, where x_0 is a PI, x_n is a PO, and x_i is a wire connecting two gates G_i and G_{i+1} in the path P. $AT(x_{i+1})$ is within the range of the summations of the arrival time of G_{i+1}'s inputs and $d(G_{i+1})$. Hence, in Fig. 5, $AT(g_1)$ is within the range of $(AT(b)+d(G_1))$ to $(AT(d)+d(G_1))$, i.e., $5 \sim 7$. Similarly, $AT(f)$ is calculated in the same manner and is within $5 \sim 11$. On the other hand, the required time of the PO is equal to the delay constraint D, and $RT(x_i)$ is the minimum required time among G_{i+1}'s fanouts subtracting $d(G_{i+1})$. Hence, in Fig. 5, $RT(f)$ is equal to D, and $RT(c)$ in G_1 is equal to $min\{RT(g_1)-d(G_1), RT(g_1')-d(G_1), RT(c) = RT(f)-d(G_2)\}$ where $RT(c)$ in G_2 is earlier computed. Since $RT(g_1) - d(G_1) = RT(f) - d(G_2) - d(G_1) = 8 - 4 - 3 = 1$ is smaller than both $RT(g_1') - d(G_1) = RT(g) - d(G_3) - d(G_1) = 8 - 2 - 3 = 3$ and $RT(c) = RT(f) - d(G_2) = 8 - 4 = 4$, $RT(c)$ is equal to $RT(g_1) - d(G_1) = 1$. The computed arrival times and required time in this example are summarized in Fig. 6.

After computing the minimal, maximal arrival times, and the required time of each node/wire in the network, we start to collect the violated paths in the network. A violated path is constructed in a DFS manner from a PI toward a PO. For a PI α, if $RT(\alpha) < AT(\alpha)$, then the PI α is collected as the partial path of a violated path. Next, for each α's fanout β, if $RT(\beta) < AT(\beta)$, then β is collected into the partial path as well. Otherwise, this path is withdrawn. The partial path continues to collect a node γ when $RT(\gamma) < AT(\gamma)$. When a PO is reached during this process, a violated path is found. The process of collecting violated paths is repeated until all PIs have been examined.

We use Fig. 6 to illustrate the process of collecting violated paths. We first start from a PI a. Since $RT(a) > AT(a)$, a is

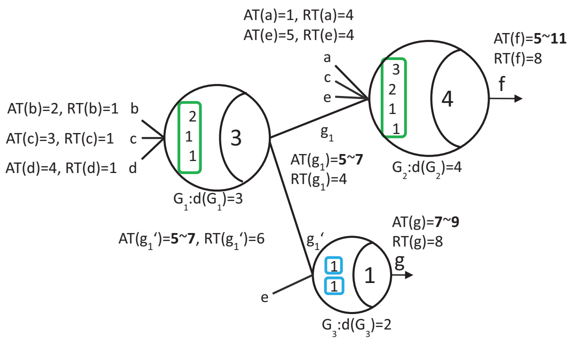

Figure 6: An illustration for collecting violated paths.

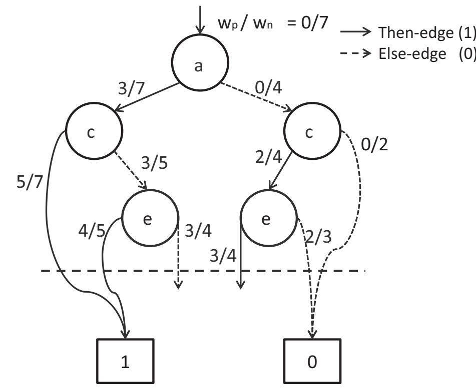

Figure 7: The sensitization BDD for G_2 in Fig. 6.

not a partial path of a violated path. Then, we check another PI b. Since $RT(b) < AT(b)$, b is a partial path of a violated path. Then, we extend this partial path to the next gate G_1 and wire g_1. Since $RT(g_1) = 4 < AT(g_1) = AT(b) + d(G_1) = 5$ from b, the extension is valid. Finally, $RT(f) = 8$ is also less than $AT(f) = 5 + 4 = 9$ from g_1. As a result, a violated path (b, G_1, g_1, G_2, f) is collected. Then, we check if there exist other violated paths from the other fanout g'_1 of gate G_1. Since $RT(g'_1) = 6 > AT(g'_1) = AT(b) + d(G_1) = 5$, (b, G_1, g'_1) is not a partial path of a violated path. For the remaining PIs, c, d, and e, we also collect the corresponding violated paths, (c, G_1, g_1, G_2, f), (d, G_1, g_1, G_2, f), (d, G_1, g'_1, G_3, g), and (e, G_2, f) in the same manner.

The overall procedure for collecting the violated paths includes arrival/required time calculation and violated path generation. The required/arrival time calculation on each node is from the POs/PIs and proceeds to its fanins/fanouts. Thus, the time complexity for this calculation is $O(V + E)$, where V is the number of LTGs and E is the number of wires in the threshold network. Next, a violated path is generated by examining the slack of each node from the PIs to the POs. Although this violated path enumeration process is theoretically time-consuming, it is practically feasible and efficient. This is because when the delay constraint is not extremely small, many paths will be pruned out without enumeration during this process. This phenomenon also can be seen in our experimental results.

B. Path Sensitization

Given a collected violated path P_j, we then try to find an assignment that satisfies the sensitization conditions for the on-input x_i on each gate G_{i+1} along P_j. According to the computed arrival time of x_i and its side-inputs, we can derive some input assignments. If the assignments are consistent, P_j is sensitizable meaning that it is a true path. Otherwise, P_j is unable to be sensitized and is a false path. The procedure of deriving the input assignments depends on the types of LTGs and its corresponding sensitization conditions. Thus, we discuss this derivation procedure for the LTGs in two categories. The first category is for Type-1 and Type-2 LTGs; the other category is for Type-3 and type-4 LTGs.

1) **Type-1 and Type-2 LTGs:** We discuss these two types of LTGs in three cases according to the precedence of the arrival time among all the inputs of an LTG.
   Case 1: x_i is the earliest input of G_{i+1}: The input assignment satisfying the sensitization condition is $x_i = cv(G_{i+1})$.
   Case 2: x_i is the latest input of G_{i+1}: x_i is either $cv(G_{i+1})$ or $ncv(G_{i+1})$ while all the side-inputs are $ncv(G_{i+1})$.
   Case 3: $AT(x_i)$ is within the range of the arrival time of the inputs in G_{i+1}: $x_i = cv(G_{i+1})$, and the earlier side-inputs are $ncv(G_{i+1})$.
2) **Type-3 and Type-4 LTGs:** We propose to construct a sensitization Binary Decision Diagram (BDD) [1] for determining if an input assignment exists for satisfying the sensitization conditions of these types of LTGs.

We use a violated path (e, G_2, f) on G_2 in Fig. 6 to demonstrate the construction of this BDD as follows. Assume that e is the on-input and we set the variable ordering of the sensitization BDD as the input arrival order of G_2, $a > c > e > g_1$. Note that, this BDD only contains the on-input e and the side-inputs that arrive not later than the on-input, i.e., a and c, since we intend to check the sensitization condition for the on-input e.

Next, we describe how to construct the sensitization BDD. Each edge in the sensitization BDD associates with two integers w_p and w_n, denoted as w_p/w_n. w_p is initialized as 0 which represents the currently accumulated weight from the stabilized inputs. w_n is initialized as the summation of all weights which represents the total amount of weights to be accumulated. For example, w_p/w_n is initialized as $0/7$ in Fig. 7 where 7 is the summation of all weights in G_2. Next, we build the *then-edge* and *else-edge* from the root node according to the assigned value, 1 or 0, of the root node. The w_p/w_n values on the edges need to be updated from the values on the coming edge of the parent node as follows. For the then-edges, w_p is updated as $(w_p + w_j)$ and w_n remains unchanged; for the else-edges, w_p remains unchanged and w_n is updated as $(w_n - w_j)$ where w_j is the input weight in the parent node. For example in Fig. 7, if input a is assigned as 1, w_p/w_n is updated as $(w_p + w_j)/w_n = (0+3)/7 = 3/7$ where w_j is the weight of a. If input a is assigned as 0, w_p/w_n is updated as $w_p/(w_n - w_j) = 0/(7 - 3) = 0/4$. The same update procedure is conducted for the other edges. For example, consider the node c on the left side, if c is assigned as 1, w_p/w_n is updated as $(3 + 2)/7 = 5/7$. If c is assigned as 0, w_p/w_n is updated as $3/(7 - 2) = 3/5$.

During the construction of the sensitization BDD, for the then-edge, once its w_p is greater than or equal to the threshold value T, the edge directly leads to the terminal 1. On the other hand, for the else-edge, once its w_n is less than T, the edge directly leads to the terminal 0. These two terminal values represent the stabilized output values. For the paths from the root node to the terminal nodes, if its last assigned input is the on-input x_i of the LTG, an input assignment satisfying the sensitization condition is found. For example in Fig. 7, there are two paths leading to the terminal 1. For the leftmost path $a \to c$, since the last assigned input is not the on-input e, e is not the dominant input under such an input assignment $(a = 1, c = 1)$. Thus, we discard this path. Next, we find another path $a \to \bar{c} \to e$ and realize the assignment $(a = 1, c = 0, e = 1)$ is responsible for sensitizing the on-input e. For the paths not leading to the terminals, like $a \to \bar{c} \to \bar{e}$, they represent that the on-input is not sensitized under this assignment. Similarly, the other sensitization path for the on-input e is $\bar{a} \to c \to \bar{e}$. For a sensitization BDD of an LTG, if there exists no sensitization path for the on-input, the path involving the on-input is a false path.

Next, we use a violated path (c, G_1, g_1, G_2, f) to demonstrate how to determine the truth or falsity of a path. Consider the on-input c in G_1 of Fig. 6, because G_1 is a Type-3 LTG, we can construct its sensitization BDD as mentioned to derive the input assignment $(b = 1, c = 1)$ for sensitizing the on-input c. After deriving the sensitized input assignments, we then check whether the assignments are consistent with the actual input values. Since inputs b, c, and d have not been assigned, we can accept this assignment. Next, we forward simulate $(b = 1, c = 1)$ and get $g_1 = g_1' = 1$ since the weight summation, $w_b + w_c = 3$, of G_1 is equal to the threshold value 3. After getting $g_1' = 1$, the PO g is set to 1 as well. As for G_2, since a and e still remain unknown, f remains unknown. Meanwhile, the corresponding arrival times are updated as $AT(g_1) = 6$, $AT(g_1') = 6$, and $AT(g) = 8$.

Next, because G_1 is not a PO, we continue to sensitize the next on-input g_1 in G_2 for this violated path. G_2 is also a Type-3 LTG, and the sensitized input assignments are, $(a = 1, c = 0, e = 0, g_1 = 1)$, $(a = 1, c = 0, e = 0, g_1 = 0)$, and $(a = 0, c = 1, e = 1, g_1 = 1)$. However, after examining these input values, we found that $c = 0$ in the first two input assignments is inconsistent with the derived assignment of $c = 1$. Thus, we select the last assignment $(a = 0, c = 1, e = 1, g_1 = 1)$. Because G_2 has reached the PO f, it means that this path is successfully sensitized under this assignment $(a = 0, b = 1, c = 1, d = -, e = 1)$ without causing any conflict.

C. Overall Flow

The overall flow of STA algorithm is shown in Fig. 4. Frst, for the given threshold logic network, its delay model, and the constraints, we explore the threshold network. During the traversal of the threshold network, we label the type of each LTG according to its input weights and threshold value, and then transform the network. After completing the transformation procedure, we compute the required time and the arrival time. By comparing the arrival time and required time, we eliminate non-violated paths during the path enumeration process. Then, we try to sensitize each violated path according to the path delay in the descending order. For each violated path, if we can derive consistent assignments in the PIs, this path is a true path. Otherwise, this violated path is a detected false path. We repeat the process for each violated path until at most K critical paths are identified.

V. Experimental Results

We implemented the proposed algorithm in C, and conducted the experiments on a 3.0 GHz Linux platform (CentOS 4.6). The benchmarks were selected from the MCNC and IWLS 2005 [12] benchmark suite in a blif format. These benchmarks were first synthesized as threshold logic networks using the tool TELS [46] with a default fanin number constraint, 6, which is the maximal number of inputs allowed in an LTG of the network. In the experiments, to demonstrate the accuracy and capability of the proposed algorithm, we compared our results against the results obtained from a timing simulator, which is an extension of the simulator provided in the synthesis tool, TELS [46].

Table I. The experimental results of timing analysis for our approach and the exhaustive simulation approach using the delay model $(1 + 0.35 \times fanin)$ for $K = 1$ and $K = 10$.

benchmark	|PI|	|LTG|	D	TLP	EX-Simulation		Ours		
					delay	T (s)	K=1		K=10
							delay	T (s)	T (s)
majority	5	1	1	2.75	2.75	<0.01	2.75	<0.01	<0.01
C17	5	4	4	5.45	5.45	<0.01	5.45	<0.01	<0.01
b1	3	8	2	3.75	3.75	<0.01	3.75	<0.01	<0.01
cm138a	6	9	3	4.45	4.45	0.03	4.45	<0.01	<0.01
cm82a	5	12	5	6.50	6.50	0.02	6.50	<0.01	0.01
cm42a	4	13	4	5.45	5.45	0.01	5.45	<0.01	<0.01
cm151a	12	14	8	9.90	9.90	2.40	9.90	<0.01	<0.01
decod	5	18	3	4.10	4.10	0.03	4.10	<0.01	<0.01
x2	10	21	5	6.85	6.85	1.14	6.85	<0.01	0.01
pm1	16	23	6	7.55	7.55	82.62	7.55	<0.01	0.02
cm163a	16	23	9	10.95	10.95	75.80	10.95	<0.01	0.02
xor5	5	25	9	10.95	10.95	0.02	10.95	<0.01	<0.01
cm162a	14	25	11	12.65	12.65	12.99	12.65	<0.01	0.01
cmb	16	25	15	16.40	16.40	73.29	16.40	0.01	0.04
cm85a	11	28	16	17.05	17.05	1.89	17.05	<0.01	0.01
cu	14	29	6	7.90	7.90	22.13	7.90	<0.01	0.01
tcon	17	32	2	3.75	3.75	128.06	3.75	<0.01	<0.01
pcle	19	38	13	14.30	14.30	668.67	14.30	<0.01	0.01
parity	16	45	12	13.60	13.60	98.10	13.60	<0.01	0.01
z4ml	7	64	7	8.60	8.60	0.30	8.60	<0.01	<0.01
sct	19	65	9	10.30	10.30	1694.56	10.30	<0.01	0.01
f51m	8	81	7	8.60	8.60	0.85	8.60	<0.01	0.01
9symml	9	131	19	20.15	20.15	2.26	20.15	0.01	0.03
alu2	10	225	38	43.90	39.45	9.96	39.45	17.64	21.69
alu4	**14**	**392**	**41**	**46.60**	**42.15**	**259.20**	**42.15**	**78.47**	**181.70**
vda	17	415	12	13.75	13.75	1806.19	13.75	0.01	0.06
ex5	8	611	25	26.70	26.70	5.21	26.70	0.03	0.18
ex1010	10	1295	32	33.20	33.20	42.63	33.20	0.06	0.59
t481	16	1311	19	20.25	20.25	5246.15	20.25	0.12	1.01
spla	16	2959	43	45.30	45.30	6233.38	45.30	0.34	3.99

In our experiments, the critical path number constraint K is set as 1 and 10. As for the delay constraint D, if it is set as a very large value, each path might not be a critical path. On the other hand, if the constraint is set as a very small value, a large amount of critical paths might be reported and hence time-consuming. Thus, we randomly simulate a small amount of patterns, i.e., 10% of the number of simulated patterns in the timing simulation approach, and adopt the obtained largest delay as the delay constraint in our experiments.

Table I summarizes the experimental results for MCNC benchmarks. Column 1 lists the benchmarks. The next two columns show the circuit information including the number of PIs (|PI|) and the number of LTGs (|LTG|). Column 4 shows the computed delay constraint D. Column 5 shows the topologically longest path delay (TLP). Columns 6 and 7 show the delay and the CPU time, measured in second, by using the simulation approach. Columns 8 to 10 show the results of the proposed STA algorithm. For example, $alu4$ benchmark has 14 PIs and 392 LTGs. The delay constraint is set as 41. The topologically longest path delay is 46.60. The exhaustive

Table II. THE EXPERIMENTAL RESULTS OF TIMING ANALYSIS FOR OUR APPROACH AND THE RANDOM SIMULATION APPROACH USING THE DELAY MODEL $(1 + 0.35 \times fanin)$ FOR $K = 1$ AND $K = 10$.

benchmark	|PI|	|LTG|	D	TLP	RD-Simulation delay	RD-Simulation T (s)	Ours K=1 delay	Ours K=1 T (s)	Ours K=10 T (s)
i1	25	25	9	10.60	10.60	7.96	10.60	<0.01	<0.01
cc	21	31	4	5.15	5.15	9.26	5.15	<0.01	<0.01
mux	21	36	18	19.50	19.50	8.67	19.50	<0.01	0.01
cm150a	21	37	11	12.30	12.30	7.53	12.30	<0.01	<0.01
pcler8	27	47	15	16.35	16.35	14.74	16.35	<0.01	0.03
cordic	23	63	17	18.10	18.10	1.26	18.10	0.01	0.02
C432	36	145	38	40.40	40.40	34.82	40.40	2.44	3.79
C880	60	232	31	33.55	32.20	57.94	33.55	1.67	1.78
C1355	41	266	30	31.75	31.75	72.36	31.75	1.14	1.44
frg1	28	281	12	13.75	13.75	71.93	13.75	0.01	0.03
C499	41	370	28	29.65	29.65	74.38	29.65	2.98	3.51
usb_phy	116	372	14	15.00	15.00	128.68	15.00	0.02	0.07
rot	135	458	28	38.45	29.80	252.30	32.90	43.92	63.74
sasc	132	627	10	11.60	11.60	214.56	11.60	0.03	0.11
C2670	233	687	35	36.90	36.90	284.25	36.90	0.63	1.06
C3540	50	772	55	59.45	56.00	200.95	59.45	9.62	16.55
i2c	146	965	21	22.55	22.55	305.07	22.55	0.06	0.20
i8	133	1191	19	20.65	20.65	541.61	20.65	2.72	7.97
C5315	182	1296	55	56.85	56.85	393.01	56.85	237.22	257.47
C6288	32	1425	160	161.75	160.10	352.57	161.75	5.01	40.60
i10	257	1527	58	68.20	62.40	723.57	65.15	686.74	727.74
systemcdes	313	2571	43	44.35	44.35	868.19	44.35	2.14	2.48
spi	273	2944	48	50.15	49.45	1275.06	49.80	262.20	388.05
aes_core	786	15351	38	39.95	39.95	7094.39	39.95	64.91	80.06
wb_conmax	1899	32317	40	41.70	41.70	25800.32	41.70	11.61	43.73
b17	**1453**	**35989**	**55**	**61.15**	**55.40**	**21288.44**	**61.15**	**17.95**	**117.75**

Table III. THE EXPERIMENTAL RESULTS OF TIMING ANALYSIS FOR OUR APPROACH AND THE EXHAUSTIVE SIMULATION APPROACH USING THE DELAY MODEL $(1 + 0.35 \times fanin + fanout)$.

benchmark	|PI|	|LTG|	D	TLP	EX-Simulation delay	EX-Simulation T (s)	Ours delay	Ours T (s)
majority	5	1	2	3.75	3.75	<0.01	3.75	<0.01
C17	5	4	9	10.45	10.45	0.01	10.45	<0.01
b1	3	8	4	5.75	5.75	<0.01	5.75	<0.01
cm138a	6	9	12	13.45	13.45	0.03	13.45	<0.01
cm82a	5	12	12	13.50	13.50	0.02	13.50	<0.01
cm42a	4	13	14	15.45	15.45	0.01	15.45	<0.01
cm151a	12	14	14	15.90	15.90	2.41	15.90	<0.01
decod	5	18	11	12.90	12.90	0.03	12.90	<0.01
x2	10	21	11	12.85	12.85	1.15	12.85	0.01
pm1	16	23	10	11.55	11.55	82.39	11.55	<0.01
cm163a	16	23	21	22.95	22.95	75.62	22.95	<0.01
xor5	5	25	16	17.60	17.60	0.02	17.60	<0.01
cm162a	14	25	31	32.65	32.65	20.99	32.65	<0.01
cmb	16	25	28	29.40	29.40	73.19	29.40	0.01
cm85a	11	28	33	34.05	34.05	1.90	34.05	<0.01
cu	14	29	9	10.90	10.90	22.17	10.90	<0.01
tcon	17	32	4	5.75	5.75	128.69	5.75	0.01
pcle	19	38	33	34.30	34.30	666.03	34.30	<0.01
parity	16	45	23	24.60	24.60	97.98	24.60	<0.01
z4ml	7	64	10	11.60	11.60	0.29	11.60	<0.01
sct	19	65	15	16.30	16.30	1695.63	16.30	<0.01
f51m	8	81	10	11.60	11.60	0.85	11.60	<0.01
9symml	9	131	31	32.10	32.10	2.26	32.10	0.01
alu2	10	225	105	124.20	106.10	9.96	106.10	15.64
alu4	**14**	**392**	**112**	**133.55**	**113.05**	**259.07**	**113.05**	**194.12**
vda	17	415	33	34.70	34.70	1806.42	34.70	0.01
ex5	8	611	73	74.30	74.30	5.21	74.30	0.02
ex1010	10	1295	130	131.45	131.45	42.68	131.45	0.06
t481	16	1311	27	28.25	28.25	5186.94	28.25	0.12
spla	16	2959	198	199.80	199.80	6245.06	199.80	0.20

simulation cost 259.20 seconds to obtain the circuit delay of 42.15 while our approach required 78.47 seconds to obtain the same delay. If we increase the critical path number from 1 to 10, the required CPU time is also increased to 181.70 seconds.

According to Table I, the delays reported from our approach and the simulation approach are the same for these benchmarks, meaning that our approach is exact. Furthermore, the required CPU time is less than that of the exhaustive simulation approach for most benchmarks. For $alu2$ benchmark, however, the required CPU time of our approach is greater than that of the simulation approach. This is because $alu2$ has a larger number of false paths, our approach needs more examinations for identifying a longest true path. On the other hand, $alu2$ benchmark has a small number of PIs such that its exhaustive simulation is affordable. Nevertheless, since the growth of exhaustive simulation time is exponential to the number of inputs, the STA approach generally requires less CPU time than the simulation approach.

Table II shows the results for the IWLS 2005 benchmarks. For some benchmarks, e.g., $b17$, in Table II, the simulation approach reported smaller delays than our approach due to the non-exhaustive, $100,000$ random patterns, simulation. Thus, the simulation approach reports a lower bound of delay for these benchmarks. According to Table II, our approach also efficiently reports exact delays for large benchmarks.

Finally, we use a different delay model in the experiments to see the delay model's impact on timing analysis. We extended the original delay model, $(1+0.35\times fanin)$, to another one, $(1+0.35\times fanin + fanout)$ that also considers the fanout number of an LTG. Table III shows the results of our approach under this extended delay model. For $alu4$ benchmark, the CPU time our approach required is increased from 78.47 to 194.12 seconds while the CPU time the exhaustive simulation approach required is almost the same. This is because the more complex delay model is, the wider diversity of path delays is, meaning that fewer paths would have the same path delay. Thus, the true paths might be identified after detecting more false paths. Nevertheless, according to Table III, the delays reported from our approach and the exhaustive simulation approach are also identical for these benchmarks. These results reveal that our STA approach is exact regardless of the circuit size and delay models.

In summary, according to the experimental results, the proposed sensitization criterion correctly and efficiently analyzed the delay of threshold logic circuits. However, the proposed path-based sensitization algorithm may not be fast enough for some benchmarks whose critical path delays are much smaller than the longest path delay. This is because the developed algorithm is to determine the falsity of the longest paths until finding the critical paths. Thus, the larger difference of delays between the longest false path and the true path is, more CPU time is required to identify the critical paths.

VI. CONCLUSION AND FUTURE WORK

In this paper, we investigate and analyze different types of threshold logic gates, and propose the first exact path sensitization criterion for threshold logic. We also develop the first STA algorithm for threshold logic circuits. Although the proposed sensitization criterion can correctly estimate the delay of threshold logic circuits, the efficiency of the developed algorithm still has room for improvement. Our future work is to study non-path-based or SAT-based STA algorithms for threshold logic circuits.

REFERENCES

[1] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Computers*, vol. C-27, pp. 509–516, 1978.

[2] M. J. Avedillo and J. M. Quintana, "A threshold logic synthesis tool for RTD circuits," in *Proc. European Symp. on Digital System Design*, 2004, pp. 624–627.

[3] M. J. Avedillo *et al.*, "Multi-threshold threshold logic circuit design using resonant tunnelling devices," *Electron. Lett.*, vol. 39, pp. 1502–1504, 2003.

[4] V. Beiu *et al.*, "VLSI implementations of threshold logic-a comprehensive survey," in *Tutorial at Int. Joint Conf. Neural Networks*, 2003.

[5] J. Benkoski *et al.*, "Timing verification using statically sensitizable paths," *IEEE Trans. CAD*, vol. 9, pp. 10 723–10 784, 1990.

[6] P. Celinski *et al.*, "Delay analysis of neuron-MOS and capacitive threshold-logic," in *Proc. Int. Conf. Electronics, Circuits and Systems*, 2000, pp. 932–935.

[7] P. Celinski *et al.*, "State of the art in CMOS threshold logic VLSI gate implementations and systems," in *Proc. Int. Conf. VLSI Circuits and Systems*, 2003, pp. 53–64.

[8] H.-C. Chen and D. H.-C. Du, "Path sensitization in critical path problem," *IEEE Trans. CAD*, vol. 12, pp. 196–207, 1993.

[9] Y.-C. Chen *et al.*, "Automated mapping for reconfigurable single-electron transistor arrays," in *Proc. DAC*, 2011, pp. 878–883.

[10] Y.-C. Chen *et al.*, "A synthesis algorithm for reconfigurable single-electron transistor arrays," *ACM Journal on Emerging Technologies in Computing System*, vol. 9, p. Article 5, 2013.

[11] C.-E. Chiang *et al.*, "On reconfigurable single-electron transistor arrays synthesis using reordering techniques," in *Proc. DATE*, 2013, pp. 1807–1812.

[12] "IWLS 2005 benchmarks," http://iwls.org/iwls2005/benchmarks.html.

[13] Y.-T. Chung and J.-H. R. Jiang, "Functional timing analysis made fast and general," in *Proc. DAC*, 2012, pp. 1055–1060.

[14] M. L. Dertouzos, *Threshold Logic: A Synthesis Approach*. M.I.T. Press, 1965.

[15] S. Devadas *et al.*, "Delay computation in combinational logic circuits: Theory and algorithms," in *Proc. ICCAD*, 1991, pp. 176–179.

[16] D. H.-C. Du *et al.*, "On the general false path problem in timing analysis," in *Proc. DAC*, 1989, pp. 555–560.

[17] S. Eachempati *et al.*, "Reconfigurable bdd-based quantum circuits," in *Proc. Int. Symp. on Nanoscale Architectures*, 2008, pp. 61–67.

[18] T. Gowda and S. Vrudhula, "Decomposition based approach for synthesis of multi-level threshold logic circuits," in *Proc. ASPDAC*, 2008, pp. 125–130.

[19] T. Gowda *et al.*, "Combinational equivalence checking for threshold logic circuits," in *Proc. GLSVLSI*, 2007, pp. 102–107.

[20] T. Gowda *et al.*, "A non-ILP based threshold logic synthesis methodology," in *Proc. IWLS*, 2007, pp. 222–229.

[21] T. Gowda *et al.*, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. CAD*, vol. 30, pp. 665–677, 2011.

[22] P. Gupta *et al.*, "Automatic test generation for combinational threshold logic networks," *IEEE Trans. VLSI Systems*, vol. 16, pp. 1035–1045, 2008.

[23] D. Hampel and R. O. Winder, "Threshold logic," *IEEE Spectrum*, vol. 8, pp. 32–39, 1971.

[24] R. B. S. Hitchcock, "Timing verification and the timing analysis program," in *Proc. DAC*, 1982, pp. 594–604.

[25] S. L. Hurst, "Sequential circuits using threshold logic gates," *Int. Journal of Electronics*, vol. 29, pp. 495–499, 1970.

[26] M. G. Johnson, "A symmetric CMOS NOR gate for high-speed applications," *IEEE. Journal of Solid-State Circuits*, vol. 23, pp. 1233–1236, 1988.

[27] Z. Kohavi, *Switching and Finite Automata Theory*. McGraw-Hill College, 1978.

[28] Y. Kukimoto and R. Brayton, "Exact required time analysis via false path detection," in *Proc. DAC*, 1997, pp. 220–225.

[29] P.-Y. Kuo *et al.*, "On rewiring and simplification for canonicity in threshold logic circuits," in *Proc. ICCAD*, 2011, pp. 396–403.

[30] Y.-M. Kuo *et al.*, "Efficient boolean characteristic function for timed automatic test pattern generation," *IEEE Trans. CAD*, vol. 28, pp. 417–425, 2009.

[31] C. Lageweg *et al.*, "A linear threshold gate implementation in single electron technology," in *Proc. Comput. Soc. Workshop VLSI*, 2001, pp. 93–98.

[32] L. A. Lev *et al.*, "A 64-b microprocessor with multimedia support," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 1227–1238, 1995.

[33] P. C. McGeer and R. K. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network," in *Proc. DAC*, 1989, pp. 561–567.

[34] P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*. Springer, 1991.

[35] S. Muroga, *Threshold Logic and its Applications*. John Wiley & Sons, 1972.

[36] S. D. Naffziger *et al.*, "The implementation of the itanium 2 microprocessor," *IEEE. Journal of Solid-State Circuits*, vol. 37, pp. 1448–1460, 2002.

[37] C. Pacha *et al.*, "Resonant tunneling device logic circuit," DortmundGerhard-Mercator University of Duisburg, Germany, Tech. Rep., 1999.

[38] M. Padure *et al.*, "Capacitive threshold logic: A designer perspective," in *Proc. Int. Semiconductor Conf.*, 1999, pp. 81–94.

[39] M. Perkowski and A. Mishchenko, "Logic synthesis for regular fabric realized in quantum dot cellular automata," *Journal of Multiple-Valued Logic and Soft Comput.*, pp. 768–773, 2004.

[40] S. Perremans *et al.*, "Static timing analysis of dynamically sensitizable path s," in *Proc. DAC*, 1989, pp. 568–573.

[41] V. Saripalli *et al.*, "Energy-delay performance of nanoscale transistors exhibiting single electron behavior and associated logic circuits," *Journal of Low Power Electronics*, vol. 6, pp. 415–428, 2010.

[42] L. G. Silva *et al.*, "Satisfiability models and algorithms for circuit delay computation," *ACM TODAES*, vol. 7, pp. 137–158, 2002.

[43] P. Venkataramani *et al.*, "Sequential circuit design in quantum-dot cellular automata," in *Proc. Nanotechnology Conf.*, 2008, pp. 534–537.

[44] R. O. Winder, "Threshold logic," Ph.D. dissertation, Princeton University, Princeton, NJ, 1962.

[45] S. H.-C. Yen *et al.*, "Efficient algorithms for extracting the k most critical paths in timing analysis," in *Proc. DAC*, 1989, pp. 649–654.

[46] R. Zhang *et al.*, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. CAD*, vol. 24, pp. 107–118, 2005.

[47] Y. Zheng *et al.*, "SAT-based equivalence checking of threshold logic designs for nanotechnologies," in *Proc. GLSVLSI*, 2008, pp. 225–230.