Rewiring for Threshold Logic Circuit Minimization

Chia-Chun Lin, Chun-Yao Wang, Yung-Chih Chen[§], Ching-Yi Huang
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chung Li, Taiwan, R.O.C.

***Abstract*—Recently, many works have been focused on synthesis, verification, and testing of threshold circuits due to the rapid development in efficient implementation of threshold logic circuits. To minimize the hardware cost of threshold circuit implementation, this paper proposes a heuristic that consists of rewiring operations and a simplification procedure. Additionally, a subset of input vectors of a gate, called critical-effect vectors, are proved to be complete for formally verifying the equivalence of two threshold logic gates, instead of the whole truth table in this paper. This achievement can accelerate the equivalence checking of two threshold logic gates. The experimental results show that the proposed heuristic can efficiently reduce the cost.**

I. Introduction

The research and development of threshold logic can be cast back to the 1960s. In 1961, an effective method to enumerating the threshold functions was proposed [28]. Later in 1962, an approximation method was proposed to determine the input weights and the threshold value of a threshold logic gate [27]. Besides, linear programming and tabulation methods were introduced to determine whether a function can be represented in threshold logic or not [26]. Although the related research on threshold logic was introduced in early days, threshold logic had a little impact on integrated circuit designs due to the lack of efficient implementation. In past decades, however, threshold logic becomes popular due to the rapid growth in nanotechnology-based devices, such as Resonant Tunneling Diodes (RTD) [2][3][12], Quantum Cellular Automata (QCA) [23], Resistance switching devices [20], Spin-based devices [1], and Single-Electron Transistor (SET) [4][29][30][19][24]. Thus, threshold logic attracts more attention than ever before.

Due to the tremendous impetus of VLSI technology, there are various implementations for threshold circuits, e.g., capacitive linear threshold gates (LTGs) or RTD-based LTGs. Accordingly, different cost functions have been proposed to evaluate the threshold circuits. In particularly, for the capacitive LTGs and RTD-based LTGs, the circuit area is proportional to the weights and threshold value of an LTG [3]. Thus, in this work, we adopt the summation of the weights and threshold value of gates in threshold networks as the cost function when synthesizing a threshold logic circuit.

On the other hand, not only the nanotechnology devices, the design automation research has also triggered a vigorous growth on threshold logic circuits. For example, the threshold logic network synthesis and fault-tolerant threshold logic designs are interesting achievements recently [13][14][31]. Static Timing Analysis of threshold logic circuits has been proposed [25]. Besides, algorithms for the equivalence checking of threshold logic circuits have been proposed [15]. Additionally, testing issue in threshold logic has also been addressed [16].

Rewiring is a logic restructuring technique that has been well developed and widely applied in the traditional Boolean circuits. It plays an important role in optimization of Boolean networks [5]-[11][18]. Recently, Kuo et al. proposed the first rewiring algorithm and a simplification procedure for threshold logic circuits [17]. Furthermore, in the simplification procedure, it proposed that if the critical-effect vectors[1] (CEVs) and their additional brother vectors[2] of two threshold logic gates are identical, the two threshold logic gates are equivalent meaning that the simplification operation is valid. However, their rewiring algorithm only focused on circuit restructuring and did not consider the cost minimization issue. Thus, in this work, we propose a heuristic to rewire the circuit for minimizing its weights and threshold value. Additionally, we also prove that the CEVs, without including their brother vectors, are sufficient to justify the equivalence of two threshold gates.

II. Preliminaries

A. Threshold logic

An LTG is a logic element whose output f is evaluated by EQ(1):

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

where w_i and T can be positive or negative numbers.

If the summation of corresponding weights w_i of inputs x_i that are assumed to be 1 in an input vector is greater than or equal to the threshold value T, the output f is 1. Otherwise, the output f is 0. The model of an LTG is shown in Fig. 1. A Boolean function realized by such an LTG is called a *threshold function*. Furthermore, a threshold function may have many different representations that are represented as a *weight-threshold vector* $\langle w_1, w_2, \ldots, w_n; T\rangle$. A network that is composed of LTGs is called a *threshold network*. Any Boolean function can be represented by a corresponding threshold network.

B. Critical-effect vectors

An LTG has a *critical-effect* if and only if there exists an assignment such that the output changes from 1 to 0 when any

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]The critical-effect vectors will be introduced in Section II.
[2]Please refer [17] for the definition of the brother vectors of CEVs.

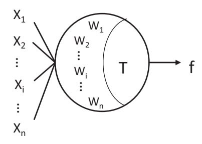
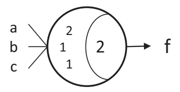


Fig. 1. An LTG model.



a	b	c	f
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Fig. 2. An LTG and its CEVs.

one of its inputs in this assignment changes from 1 to 0. An input assignment that satisfies the requirement of the critical-effect for an LTG is called a *critical-effect vector (CEV)*. For example, in Fig. 2, input assignments 011 and 100 are the CEVs of LTG $\langle 2, 1, 1; 2\rangle$. This is because changing any 1 to 0 in these assignments will also change the output from 1 to 0.

EQ(2) shows the sufficient condition of an input assignment being a CEV, where n is the number of inputs in an LTG.

$$\sum_{i=1}^{n} x_i w_i = T \tag{2}$$

Note that although in this example, the two CEVs both satisfy EQ(2), the CEVs of an LTG do not always satisfy this equation.

Since the concept of CEV plays an important role in the proposed simplification flow, here we briefly introduce how to find the CEVs of an LTG. First, the weights are sorted in a descending order. By the definition of a CEV, we know that when any input in this assignment changes from 1 to 0, the output changes from 1 to 0. Thus, in the procedure, we iteratively decrease the threshold value by the largest input weight and check the threshold value. If the threshold value after the decrement becomes less than or equal to 0, the procedure is terminated, and the threshold value is restored for the next CEV. In each iteration of the procedure, when we reduce the threshold value by w_i, the corresponding input x_i is set to 1; otherwise, it is set to 0. All the input assignments obtained in this procedure are the CEVs of this threshold gate.

In the example of Fig. 2, we first decrease the threshold value by 2 after setting $a = 1$, then the threshold value becomes 0 and we get a CEV = 100. Next, we set $\{a = 0, b = 1, c = 1\}$ and the threshold value is also decreased from 2 to 0. Hence, 011 is another CEV of an LTG. The proposed heuristic for finding CEVs is quite affordable. This is because in practice, the number of CEVs of an LTG is not large compared to the number of all input patterns.

C. Weight transformation

By the definition of LTG, the weights and threshold value can be positive or negative integers. In this work and the proposed theorems, however, we assume that the weights and threshold value are positive integers for facilitating the analysis of the threshold network. A threshold gate with negative weights or threshold value can be transformed to the one with positive weights and threshold value by the method detailed in [22].

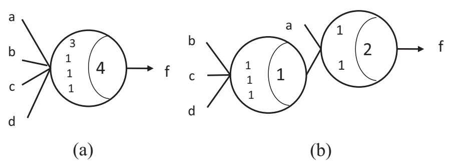


Fig. 3. (a) An LTG. (b) A more efficient implementation.

III. Cost minimization algorithm

The most commonly used approach to synthesizing a threshold network is the ILP-based algorithm [31]. Although this algorithm is efficient to synthesize threshold networks for arbitrary functions, it actually achieves a local optimum of weights and threshold value under a given fanin number constraint[3]. Additionally, the algorithm tries to use fewer numbers of LTG to express a given function, which may increase weights or threshold value.

For example, given a function $f = ab + ac + ad$. Fig. 3(a) is a corresponding threshold gate synthesized by the ILP-based algorithm. In contrast with Fig. 3(a), Fig. 3(b) is another representation of the function f which is synthesized by the rewiring algorithm. We can see that the summation of all the weights and threshold value of Fig. 3(a) is 10, while that of Fig. 3(b) is only 8. As a result, a synthesized threshold network with fewer LTGs does not always achieve the globally minimal cost. In the following paragraphs, we first review some terminology proposed in [17].

Definition 1: An LTG is *useless* if and only if it outputs zero for all input combinations.

Definition 2: An input in an LTG is *critical* if and only if this LTG will become useless after removing this input.

A. Overview

Fig. 4 gives an overview of our algorithm. The input is a threshold network, and the output is a rewired threshold network with the reduced cost. We first identify a target wire such that the cost can be reduced after removing the target wire and adding the rectification network. After the rewiring operations, the appearance of some LTGs may be changed. Thus, we perform the simplification procedure to minimize the weights and threshold value.

B. Target wire selection

In this subsection, we introduce how to choose the target wire such that the cost will be reduced after removing the target wire and adding the rectification network. We summarize two cases that are capable of achieving this cost reduction potentially.

Case 1: An input having the weight that equals the threshold value: In this situation, if the input of this weight is 1 in a vector, the output f is 1 without considering the values of other inputs. It means that we can split this input from the threshold gate since it independently determines the output value. If such inputs are not unique in an LTG, we simultaneously remove them. Then, we connect the remaining threshold gate to the removed target wire with an OR gate

[3]The fanin number constraint is a limitation that the input number of every LTG in the threshold circuit cannot exceed.

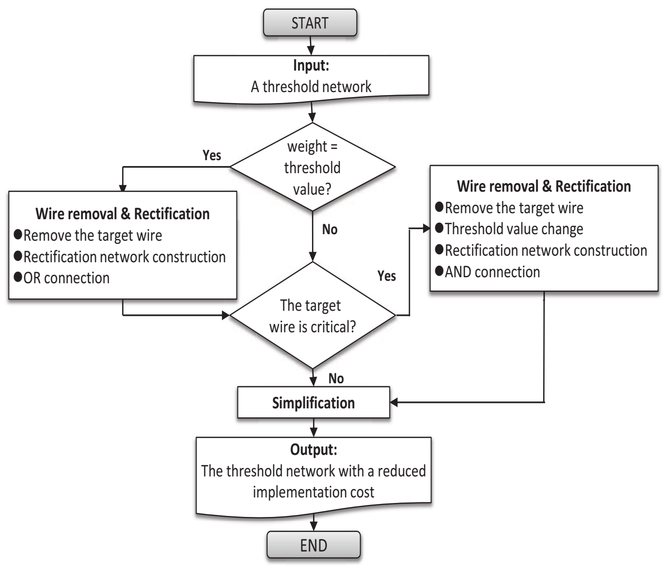

START
Input:
A threshold network
weight = threshold value?
Yes
No
Wire removal & Rectification
●Remove the target wire
●Rectification network construction
●OR connection
The target wire is critical?
Yes
No
Wire removal & Rectification
●Remove the target wire
●Threshold value change
●Rectification network construction
●AND connection
Simplification
Output:
The threshold network with a reduced implementation cost
END

Fig. 4. Our overall flow.

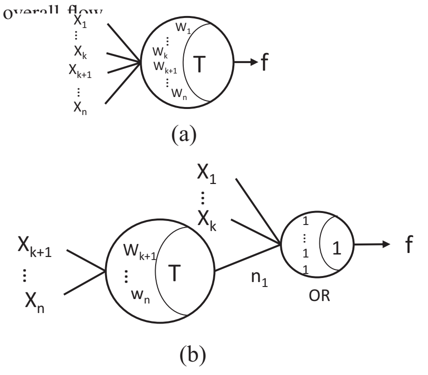


Fig. 5. Case 1 (a) The LTG before rewiring. (b) The resultant threshold network after rewiring.

at its transitive fanout cone. Although the functionality is not changed after the rewiring operation [17], we have to further ensure that this rewiring operation *does* reduce the cost. Thus, we propose Theorem 1 that states the condition for having a cost reduction.

Theorem 1. *Given an n-input ($x_1 \sim x_n$) LTG with k symmetric inputs $x_1 \sim x_k$; $k \geq 1$: $G_1 = \langle w_1, \ldots, w_k, w_{k+1}, \ldots, w_n; T\rangle$, $w_1 = w_2 = \ldots = w_k = T$, the rewired threshold network has a cost reduction if and only if $k(T-1) > 2$.*

Proof: ($\Rightarrow$) Given such an LTG $G_1 = \langle w_1, \ldots, w_k, w_{k+1}, \ldots, w_n; T\rangle$, as shown in Fig. 5(a). According to the rewiring operation, the resultant network is as shown in Fig. 5(b). To ensure the cost of Fig. 5(b) is less than that of Fig. 5(a), we have $w_1 + \ldots + w_k + w_{k+1} + \ldots + w_n + T > w_{k+1} + \ldots + w_n + T + (k+1) + 1$. Because $w_1 = w_2 = \ldots = w_k = T$, the inequality can be rewritten as $T + T + \ldots + T + (w_{k+1} + \ldots + w_n + T) > (w_{k+1} + \ldots + w_n + T) + (k+1) + 1$. After refining this inequality, we have $kT > k+2$, or $k(T-1) > 2$.

($\Leftarrow$) The proof can be done by reversing the proof in ($\Rightarrow$). $\square$

Case 2: A critical input: According to the definition of critical input we mentioned before, the LTG will become useless, i.e., always produce 0, after removing the critical input. However, a useless gate is not allowed in threshold circuits since it actually is a constant 0. To prevent this LTG from being a useless gate, we have to reduce the threshold value of this LTG by the weight of critical input accordingly after removing the critical input. After removing the target wire x_t and the corresponding weight w_t, we construct the rectification network using the target wire x_t only. Finally, we connect the remaining LTG to the rectification network with an AND gate at its transitive fanout cone [17]. We also propose Theorem 2 for Case 2 to state the condition that makes the cost reduced.

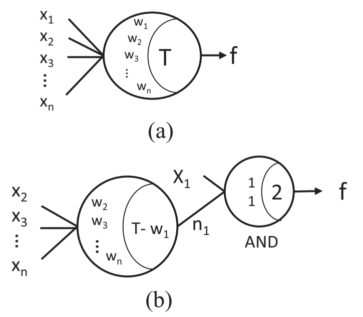


Fig. 6. Case 2 (a) The LTG before rewiring. (b) The resultant threshold network after rewiring.

Theorem 2. *Given an n-input ($x_1 \sim x_n$) LTG with the critical input x_1 : $G_1 = \langle w_1, w_2, w_3, \ldots, w_n; T\rangle$, the rewired threshold network has a cost reduction if and only if $w_1 > 2$.*

Proof: ($\Rightarrow$) Given such an n-input LTG $G_1 = \langle w_1, w_2, w_3, \ldots, w_n; T\rangle$, as shown in Fig. 6(a). According to the rewiring operation, the resultant threshold network is as shown in Fig. 6(b). To ensure the cost of Fig. 6(b) is less than that of Fig. 6(a), we have $w_1 + \mathbf{w_2} + \mathbf{w_3} + \ldots + \mathbf{w_n} + \mathbf{T} > \mathbf{w_2} + \mathbf{w_3} + \ldots + \mathbf{w_n} + (\mathbf{T} - w_1) + 1 + 1 + 2$. After simplifying this inequality, we have $2w_1 > 4$, or $w_1 > 2$.

($\Leftarrow$) The proof can be done by reversing the proof in ($\Rightarrow$). $\square$

IV. Simplification

A. Functional equivalence of two LTGs

A threshold function has different LTG representations. For example, given two LTGs $\langle 2, 1; 3\rangle$ and $\langle 1, 1; 2\rangle$, we can recognize that both LTGs represent the same function $f(a, b) = ab$, because they both produce 1 if and only if $\{a = 1, b = 1\}$.

In this subsection, we propose Theorem 3, which can efficiently verify the functional equivalence of two LTGs with different appearances.

Theorem 3. *Given two LTGs with positive weights and threshold values, they are functionally equivalent if and only if they have the same CEVs.*

Proof: Given two LTGs G_1 and G_2 with n inputs, assume CEV_{G1} and CEV_{G2} denote the CEVs of G_1 and G_2, respectively. $\ominus_i^n$ denotes n-bit vectors in which the number of 1 is i. For example, $\ominus_1^4 = \{0001, 0010, 0100, 1000\}$. Thus, $\cup_{i=0}^{n} \ominus_i^n$

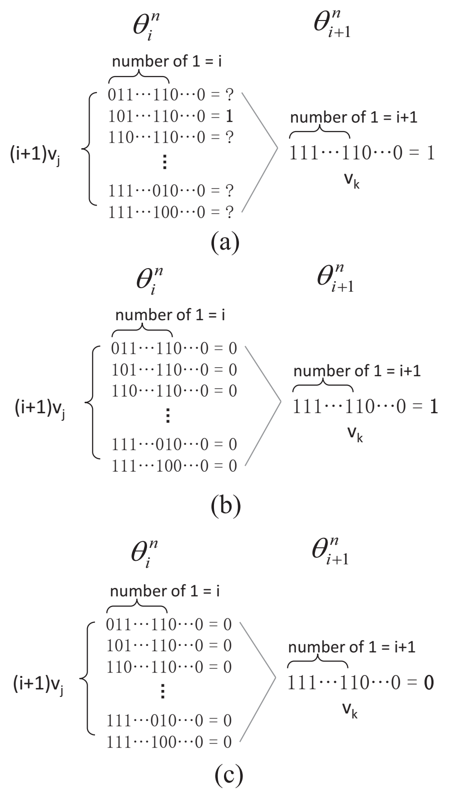


Fig. 7. (a) Case 1 in Theorem 3. (b) Case 2a in Theorem 3. (c) Case 2b in Theorem 3.

represents the whole input space. We also use the notation $G_1(v_j)$ to represent the output of gate G_1 under the vector v_j.

($\Rightarrow$) If G_1 and G_2 are functionally equivalent, $G_1(v_i) = G_2(v_i)$ for any input vector v_i. Thus, their CEVs are also identical.

($\Leftarrow$) Given $CEV_{G1} = CEV_{G2}$, we use the mathematical induction to prove the functional equivalence of G_1 and G_2. That is, we would like to show that $G_1(\ominus_i^n) = G_2(\ominus_i^n)$ for $i = 0 \sim n$.

1) **Base case:** For $i = 0$: Since $\ominus_0^n$ contains only the all-0 vector, the weighted summation of G_1 and G_2 under $\ominus_0^n$ is zero. Hence, $G_1(\ominus_0^n) = G_2(\ominus_0^n) = 0$.
2) **Induction hypothesis:** For $i = 1 \sim n$: We would like to show that if $G_1(\ominus_i^n) = G_2(\ominus_i^n)$, then $G_1(\ominus_{i+1}^n) = G_2(\ominus_{i+1}^n)$ in this step. Assume $hd(v_j, v_k)$ denotes the hamming distance between vectors v_j and v_k. We observed that any $v_k \in \ominus_{i+1}^n$, there exist $(i+1)$ $v_j \in \ominus_i^n$ such that $hd(v_j, v_k) = 1$. In this proof segment, we discuss each $v_k \in \ominus_{i+1}^n$ and summarize that for each v_k and all its $(i+1)$ v_j, they will fall into one of the following two cases.
   a) **Case 1:** (Refer Fig. 7(a), $\exists G_1(v_j) = 1$, and $G_1(v_k) = 1$) If there exists a v_j with $hd(v_j, v_k) = 1$ and $G_1(v_j) = 1$, then $G_1(v_k) = 1$ by the definition of threshold logic. This is because for such a v_j, the CEVs of bit assumed to be 1 is the subset of that of v_k. That is, it is impossible that for any v_j, $G_1(v_j) = 1$, but $G_1(v_k) = 0$. Thus, because we are given $G_1(\ominus_i^n) = G_2(\ominus_i^n)$, that means $G_1(v_j) = G_2(v_j)$ for all v_j, we can conclude that $G_2(v_k) = 1$, and $G_1(v_k) = G_2(v_k)$ in this case.
   b) **Case 2:** When $G_1(v_j) = 0$ for all $(i+1)$ v_j, $G_1(v_k)$ could have two outcomes, either $G_1(v_k) = 1$ or $G_1(v_k) = 0$. Thus, we further divide this case into two subcases.
      i) **Case 2a:** (Refer Fig. 7(b), $\forall G_1(v_j) = 0$, and $G_1(v_k) = 1$) If for all v_j, $G_1(v_j) = 0$, and $G_1(v_k) = 1$, then $v_k \in CEV_{G1}$ by the definition of CEV. Since $CEV_{G1} = CEV_{G2}$, we know that $v_k \in CEV_{G2}$. Thus, because we are given $G_1(\ominus_i^n) = G_2(\ominus_i^n)$, that means $G_1(v_j) = G_2(v_j) = 0$ for all v_j, we can conclude that $G_2(v_k) = 1$, and $G_1(v_k) = G_2(v_k)$ in this subcase.
      ii) **Case 2b:** (Refer Fig. 7(c), $\forall G_1(v_j) = 0$, and $G_1(v_k) = 0$) If for all v_j, $G_1(v_j) = 0$, and $G_1(v_k) = 0$, then v_k is not a CEV, i.e., $v_k \notin CEV_{G1}$. Since $CEV_{G1} = CEV_{G2}$, $v_k \notin CEV_{G2}$. As a result, $G_2(v_k) = 0$, and $G_1(v_k) = G_2(v_k)$ in this subcase.

   Cases 1 and 2 contain all situations of $v_j \in \ominus_i^n$ and $v_k \in \ominus_{i+1}^n$. Thus, when $G_1(\ominus_i^n) = G_2(\ominus_i^n)$, we conclude that $G_1(\ominus_{i+1}^n) = G_2(\ominus_{i+1}^n)$.
3) By 1) and 2) and mathematical induction, $G_1(\ominus_i^n) = G_2(\ominus_i^n)$ for $i = 0 \sim n$. □

According to Theorem 3, we can only examine the CEVs rather than the whole truth table of these two LTGs for checking their equivalence. Thus, with this method, the efficiency of LTG simplification can be boosted.

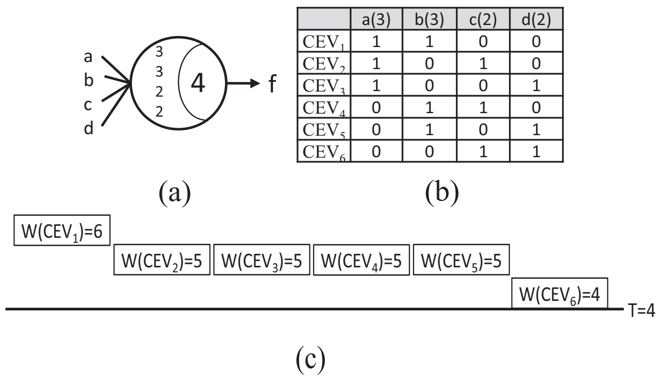
B. Simplification overview

In this subsection, we give an overview of the proposed simplification procedure. According to Theorem 3, we know that given two LTGs, they are functionally equivalent if and only if they have the same CEVs. That is, if we can ensure the CEVs intact after the simplification, this simplification is valid and we can obtain an LTG with smaller weights or threshold value. Let us use Fig. 8 as an example to illustrate this idea. Fig. 8(a) is a 4-input LTG that needs to be simplified. Its 6 CEVs are shown in Fig. 8(b). We denote the summation of weights with respect to the assignment 1 in a CEV_i as W(CEV_i). For example, W(CEV_1) of CEV_1 = 1100 is $3+3=6$. From the definition of CEV, we know that each W(CEV_i) is greater than or equal to the threshold value. Hence, we can draw a figure like Fig. 8(c) where all W(CEV_i) are above the threshold value line, $T=4$.

In the simplification process of an LTG, we try to minimize its weights and the threshold value without changing its functionality. Hence, the main idea is to keep all CEVs being above the threshold value line after the simplification.

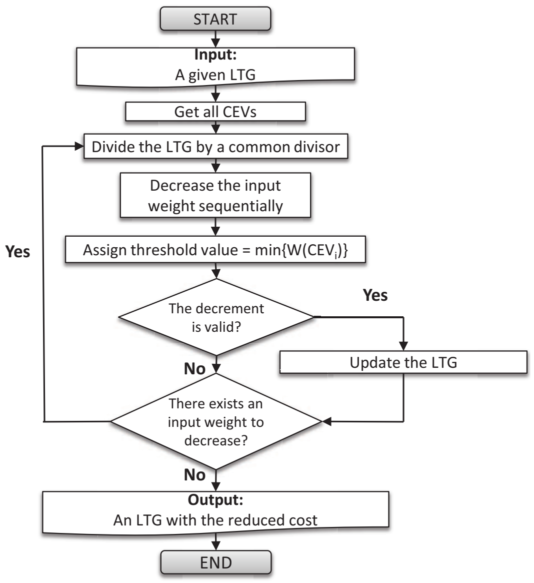
C. Simplification flow

The proposed simplification flow is shown in Fig. 9. In the beginning, we derive all the CEVs of this LTG. Second, a



	a(3)	b(3)	c(2)	d(2)
CEV$_1$	1	1	0	0
CEV$_2$	1	0	1	0
CEV$_3$	1	0	0	1
CEV$_4$	0	1	1	0
CEV$_5$	0	1	0	1
CEV$_6$	0	0	1	1

Fig. 8. (a) An LTG. (b)Its CEVs. (c)The relationship between W(CEV_i) and T.



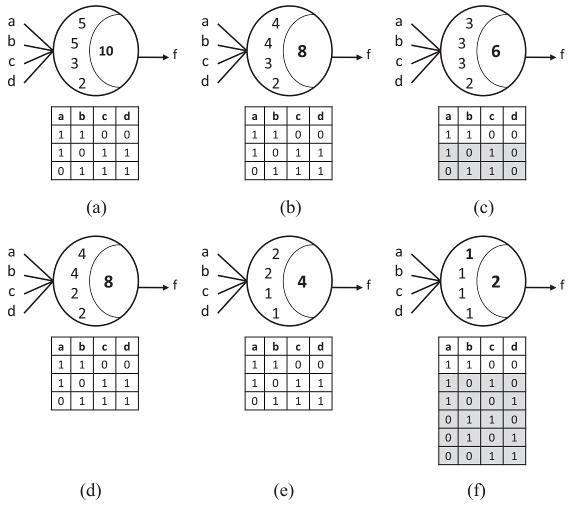
START
Input:
A given LTG
Get all CEVs
Divide the LTG by a common divisor
Decrease the input weight sequentially
Assign threshold value = min{W(CEV$_i$)}
Yes
The decrement is valid?
Yes
Update the LTG
No
There exists an input weight to decrease?
No
Output:
An LTG with the reduced cost
END

Fig. 9. Our simplification flow.

larger-than-1 common divisor, if any, divides the weights and the threshold value. Then we decrease one input weight[4]. Next, we assign the threshold value as the smallest W(CEV_i) of this LTG. After these steps, we have to check if the decrement is valid or not by comparing the CEVs of the original LTG and the new LTG according to Theorem 3. If the decrement is valid, we update the LTG. The simplification procedure is terminated when no more input weights can be decreased.

Here, we use an example to demonstrate our simplification algorithm. Given an LTG $\langle 5, 5, 3, 2; 10\rangle$, we get its CEVs 1100, 1011 and 0111, as shown in Fig. 10(a). Since there is no common divisor among the weights and threshold value, we perform the next step. For each iteration of weight decreasing operation, we sequentially decrease the input weight. In the beginning, we decrease the weight of input a from 5 to 4. However, because the input b has the same weight, we decrease it simultaneously. Next we determine the threshold value from W(CEV). The corresponding W(CEV) of CEV 1100, 1011, and 0111 are 8, 9, and 9, respectively. Thus, we

[4]We simultaneously decrease the weights of symmetric inputs. This is because if the weights of the symmetric inputs become different after the simplification, the new LTG is nonequivalent to the original one.



(a)

a	b	c	d
1	1	0	0
1	0	1	1
0	1	1	1

(b)

a	b	c	d
1	1	0	0
1	0	1	1
0	1	1	1

(c)

a	b	c	d
1	1	0	0
1	0	1	0
0	1	1	0

(d)

a	b	c	d
1	1	0	0
1	0	1	1
0	1	1	1

(e)

a	b	c	d
1	1	0	0
1	0	1	1
0	1	1	1

(f)

a	b	c	d
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1

Fig. 10. An example for simplification.

assign the threshold value as the minimal W(CEV), 8. After the decreasing step, we have to check if the decrement is valid or not by comparing their CEVs. Since the CEVs of these two LTG are the same, the decrement is valid and the updated LTG is $\langle 4, 4, 3, 2; 8\rangle$, as shown in Fig. 10(b).

Next, the decrement operation for input b is performed. Also, the inputs a and b are symmetric inputs, thus, we decrease them at the same time, and the corresponding W(CEV) of CEV 1100, 1011, and 0111 are 6, 8, and 8, respectively. We update the threshold value by the minimal W(CEV), 6. However, the decrement is invalid because the CEVs are changed, as shown in Fig. 10(c).

Now we decrease input c from Fig. 10(b). Again, we find that the threshold value is 8. This is a valid decrement because the CEVs are intact, as shown in Fig. 10(d).

In the fourth iteration, we can find a common divisor, 2, among the weights and threshold value. Hence, a new representation $\langle 2, 2, 1, 1; 4\rangle$ is obtained, as shown in Fig. 10(e). Since the weight of input d is already the minimal positive integer, the weight cannot be decreased anymore.

In the last iteration, we decrease the weights of inputs a and b. However, the resultant representation $\langle 1, 1, 1, 1; 2\rangle$ is invalid due to different CEVs. Since there is no more weight can be decreased, we terminate the procedure. The final LTG is as shown in Fig. 10(e), and the implementation cost is reduced from 25 to 10.

V. Experimental Results

We implemented the proposed heuristic in C++ language. The experiments were conducted on a 3.0 GHz Linux platform (CentOS 4.6). The benchmarks are from IWLS 2005 [32] and MCNC, and each benchmark was initially synthesized as a threshold network [31], using $|\text{PI}|$ as the fanin number constraint.

In the experiment, we demonstrate the capability of our heuristic on the cost function reduction. Table I summarizes the experimental results. Column 1 shows the names of the benchmarks. Column 2 lists the number of LTGs of the benchmarks. Column 3 lists the original cost of the benchmarks.

TABLE I. THE EXPERIMENTAL RESULTS OF OUR APPROACH.

benchmark	|LTG|	ori_cost	ours			ratio(%)
			|case 1|	|case 2|	cost	
usb_phy	285	1586	4	20	1498	5.55
C1908	295	1671	1	8	1631	2.39
rot	394	1960	6	26	1878	4.18
alu4	380	1986	4	7	1934	2.62
apex6	428	2079	4	21	2007	3.46
C1355	349	2102	2	0	2098	0.19
x3	438	2170	2	54	2054	5.35
k2	379	2228	16	11	2097	5.88
t481	324	2237	23	52	2037	8.94
simple_spi	576	2626	9	24	2540	3.27
pci_spoci_ctrl	581	3254	24	34	3127	3.90
i2c	**510**	**3268**	**26**	**35**	**2867**	**12.27**
frg2	525	3299	66	72	2977	9.76
dalu	841	3644	6	8	3608	0.99
pair	766	4057	9	36	3945	2.76
C5315	1142	4661	0	5	4651	0.21
C7552	1633	6468	9	15	6412	0.87
s9234	1058	7056	19	95	6415	9.08
i10	1354	7490	41	142	6888	8.04
des	1906	8906	11	11	8860	0.52
s13207	1853	9542	18	83	9221	3.36
C6288	1833	9892	12	12	9844	0.49
systemcdes	2134	11677	19	224	11139	4.61
spi	1911	12004	74	142	11184	6.83

Columns 4 and 5 show the numbers of rewired LTGs in case 1 and case 2, respectively. Column 6 lists the reduced cost of the benchmarks. Column 7shows the cost reduction ratio. For example, the *i2c* benchmark has 510 LTGs and the original cost is 3268. The cost was reduced to 2867 and 26 LTGs in case 1 and 35 LTGs in case 2 were rewired by the heuristic. The cost reduction ratio is 12.27%.

According to Table I, we realize that this cost reduction strongly depends on circuit structures. For some circuit, e.g., *C6288*, the reduction is very little due to fewer profitable structures. However, some circuit like *i2c* has up to 12% cost reduction. On the other hand, the required CPU time for each circuit is less than 1 second. Note that we do not compare with any previous work, since the objectives of these works are quite different from ours.

VI. CONCLUSION

Different from the prior work that minimizes the number of LTGs for synthesizing a threshold network, this paper considers the objective of cost function reduction instead. A simplification procedure that includes an efficient equivalence checking operation of two LTGs is also proposed. As the threshold logic is becoming popular, the proposed method will benefit the realization of digital designs in threshold logic.

REFERENCES

[1] C. Augustine et al., "Low-power functionality enhanced computation architecture using spin-based devices," in *Proc. Int. Symp. Nanoscale Architecture*, 2011, pp. 129-136.

[2] M. J. Avedillo et al., "Multi-Threshold Threshold Logic Circuit Design Using Resonant Tunnelling Devices," *Electron. Lett.*, vol. 39, no. 21, Oct. 2003, pp. 1502-1504.

[3] V. Beiu et al., "VLSI Implementations of Threshold Logic-a Comprehensive Survey," *IEEE Trans. on Neural Networks*, 2003, pp. 1217-1243.

[4] C. E. Chiang et al., "On Reconfigurable Single-Electron Transistor Arrays Synthesis Using Reordering Techniques," *in Proc. DATE*, 2013, pp. 1807-1812.

[5] S. C. Chang et al., "Circuit Optimization by Rewiring," *IEEE Trans. on Computers*, 1999, pp. 962-970.

[6] S. C. Chang et al., "Fast Boolean Optimization by Rewiring," in *Proc. ICCAD*, 1996, pp. 262-269.

[7] Y. C. Chen et al., "Node Addition and Removal in the Presence of Don't Cares," in *Proc. DAC*, 2010, pp. 505-510.

[8] Y. C. Chen et al., "Fast Detection of Node Mergers Using Logic Implications," in *Proc. ICCAD*, 2009, pp. 785-788.

[9] Y. C. Chen et al., "An Improved Approach for Alternative Wire Identification," in *Proc. ICCD*, 2005, pp. 711-716.

[10] Y. C. Chen et al., "Fast Node Merging with Dont Cares Using Logic Implications," *IEEE Trans. on Computer-Aided Design*, 2010, pp. 1827-1832.

[11] Y. C. Chen et al., "Logic Restructuring Using Node Addition and Removal," *IEEE Trans. on Computer-Aided Design*, 2012, pp. 260-270.

[12] D. Goldharber-Gordon et al.. "Overview of Nanoelectronic Devices," in *Proc. IEEE*, 1997, pp. 521-540.

[13] M. K. Goparaju, "A Fault Tolerant Design Methodology for Threshold Logic Gates and Its Optimizations," in *Proc. Int. Symposium on Quality Electronic Design*, 2007, pp. 420-425.

[14] T. Gowda et al., "Identification of Threshold Functions and Synthesis of Threshold Networks," *IEEE Trans. on Computer-Aided Design*, 2011, pp. 665-677.

[15] T. Gowda et al., "Combinational Equivalence Checking for Threshold Logic Circuits," in *Proc. Great Lake Symp. VLSI*, 2007, pp. 102-107.

[16] P. Gupta et al., "Automatic Test Generation for Combinational Threshold Logic Networks," *IEEE Trans. Very Large Scale Integration Systems*, 2008, pp. 1035-1045.

[17] P. Y. Kuo et al., "On Rewiring and Simplification for Canonicity in Threshold Logic Circuits," in *Proc. ICCAD*, 2011, pp. 396-403.

[18] C. C. Lin et al., "Rewiring Using Irredundancy Removal and Addition," in *Proc. DATE*, 2009, pp. 324-327.

[19] C. Lageweg et al., "A Linear Threshold Gate Implementation in Single Electron Technology," in *Proc. IEEE Computer Society Workshop on VLSI*, 2001, pp. 93-98.

[20] A. Pappachen et al., "Resistive Threshold Logic," *IEEE Trans. Very Large Scale Integration Systems*, 2013, pp. 1-5.

[21] K. Maezawa et al., "High-Speed and Low-Power Operation of A Resonant Tunneling Logic Gate MOBILE," *IEEE Eletron Device Letters*, 1998, vol. 19, pp. 80-82.

[22] S. Muroga, *"Threshold Logic and its Applications"*. New York, NY: John Wiley, 1971.

[23] M. Perkowski et al., "Logic Synthesis for Regular Fabric Realized in Quantum Dot Cellular Automata," in *Proc. Int. J. Multiple-Valued Logic and Soft Comput.*, 2004, pp. 768-773.

[24] V. Saripalli et al., "Energy-Delay Performance of Nanoscale Transistors Exhibiting Single Electron Behavior and Associated Logic Circuits," *J. Low Power Electron*, 2010, pp. 415-428.

[25] C. K. Tsai et al., "Sensitization Criterion for Threshold Logic Circuits and its Application," in *Proc. ICCAD*, 2013, pp. 226-233.

[26] R. O. Winder, "Enumeration of Seven-Argument Threshold Functions," *IEEE Trans. on Electronic Computers*, 1965, pp. 315-325.

[27] R. O. Winder, *"Threshold Logic,"* Ph.D. dissertation, Princeton University, Princeton, NJ, 1962.

[28] R. O. Winder, "Single Stage Threshold Logic," *Switching Circuit Theory and Logical Design*, 1961, pp. 321-332.

[29] C. Y. Wang et al., "Automated Mapping for Reconfigurable Single-Electron Transistor Arrays," in *Proc. DAC*, 2011, pp. 878-883.

[30] C. Y. Wang et al., "A Synthesis Algorithm for Reconfigurable Single-Electron Transistor Arrays," *ACM Journal on Emerging Technologies in Computing System*, 2013, Vol. 9, No. 1, Article 5.

[31] R. Zhang et al., "Synthesis and Optimization of Threshold Logic Networks with Application to Nanotechnologies," in *Proc. DATE*, 2004, pp. 904-909.

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