In&Out: Restructuring for Threshold Logic Network Optimization

Chia-Chun Lin, Chiao-Wei Huang, Chun-Yao Wang, Yung-Chih Chen[§]
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chungli, Taiwan, R.O.C.
d105062801@cs.nthu.edu.tw, laoday025@gmail.com, wcyao@cs.nthu.edu.tw, ycchen.cse@saturn.yzu.edu.tw

Abstract

Threshold logic attracts a lot of attention recently due to nanotechnology advances on its physical implementation. Hence, many previous works have focused on the synthesis of threshold logic networks from the Boolean functions. In this paper, we propose a tool, In&Out, which optimizes a threshold logic network from the physical implementation perspective. In&Out consists of two techniques: **Add-in** and **Remove-out**. The Add-in is to transform a threshold logic network into an implementation friendly network while the Remove-out is to remove redundant gates from the network. We conducted the experiments on a set of IWLS 2005 benchmarks. The experimental results show that the proposed tool can reduce the implementation cost of threshold networks up to 10% for the benchmarks.

Keywords

Threshold logic, optimization

1. Introduction

The research on threshold logic can be cast back to the 1960s. In 1961, an effective approach to enumerating the threshold functions was proposed [31]. In 1962, an approximation method was proposed to determine the input weights and the threshold value of a threshold logic gate [32]. Later, linear programming and tabulation methods were proposed to determine whether a function can be represented in one threshold logic gate or not [33]. Furthermore, for combinational and sequential threshold logic circuits, many different hardware implementations have been proposed in the early days [18] [19]. However, due to the lack of effective hardware realization, the research of design automation methods for threshold logic was limited as compared with that of Boolean logic. Recently, CMOS implementations of threshold logic gates have been developed and motivate the re-investigation of threshold logic. On the other hand, threshold logic gates are the basic nodes of artificial neural network [2], which is the underlying platform of machine learning [20]. Furthermore, with the advances of nanotechnologies, many nanoscale devices had been developed, such as resonant tunneling diode [1] [26], single-electron transistor [6] [7] [9] [12] [22] [28], and quantum cellular automata [27] [30], which also provide promising and efficient implementations of threshold logic gates. It had been shown that threshold logic circuits are more area-efficient than the traditional Boolean CMOS design style on arithmetic units, like adders and counters [5] [10] [34].

With the advances of implementing threshold logic devices, the design automation research on threshold logic has a rapid development. The multi-level synthesis methodologies of threshold logic network have been proposed [8] [14] [15] [16] [25] [34]. A static timing analysis for threshold logic circuits has been proposed [29]. The verification and equivalence checking on threshold logic networks have also been proposed [17] [35]. Moreover, the testing issue for the threshold logic networks has been addressed [13].

A *Linear Threshold Gate* (LTG) is a logic gate with n binary inputs, $x_1, x_2, \ldots, x_n$, and one binary output f. The representation of an LTG is shown in Fig. 1. The elements of an LTG are weights, $w_1, w_2, \ldots, w_n$, which are associated with its inputs $x_1, x_2, \ldots, x_n$, and a threshold value T. These weights can be positive or negative integers. The output f of an LTG is defined as EQ(1).

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

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. A *threshold function* is a Boolean function that can be realized by a single LTG. Compactness is one of the advantages of threshold logic. It means that we can represent a Boolean function by using a fewer threshold logic gates. For instance, $f = x_1 + (x_2(x_3 + x_4 + x_5 x_6))$ in Fig. 2(a) can be represented by a single LTG as shown in Fig. 2(b).

In the implementation of threshold logic circuits, there are many factors that need to be considered, e.g., the number of input variables of an LTG, the maximum number of fanout of an LTG, and the ratio between the maximum weight and the minimum weight of an LTG. Some factors are even the restrictions for efficient hardware implementations of threshold logic circuits [11]. Furthermore, the general objectives of synthesizing a threshold logic network are the minimization of summation of weights and threshold value, the minimization of gate count, and the minimization of level. The first two objectives are related to the area of circuits while the last one is related to the delay of the circuits. However, these objectives usually are the tradeoffs in the design consideration [4].

In the multi-level synthesis and optimization approaches for threshold logic networks, the authors usually need cost functions for guiding the optimization process and evaluating the quality of the resultant threshold logic networks. Unfortunately, some previous works [8] [25] only used the gate count of a circuit as the cost function while others only used the summation of weights and threshold values in the whole circuit as the cost function [23]. In fact, considering only one of them as the cost function is improper, this is because the resultant threshold logic networks will be extremely biased. That is, when using the gate count as the cost function, the network

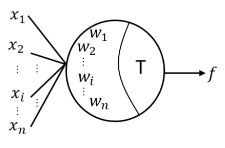


Fig. 1. An LTG model.

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3, MOST 105-2221-E-155-068, and NSC 102-2221-E-007-140-MY3.

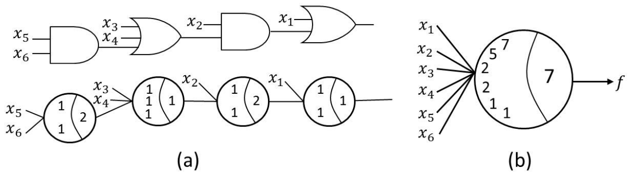


Fig. 2. The Boolean function $f = x_1 + (x_2(x_3 + x_4 + x_5x_6))$ representing in (a) Boolean logic and its corresponding threshold logic. (b) A single threshold logic gate.

with a fewer number of LTG is obtained, even the LTGs might have more input variables, and extremely large weights and threshold values. This situation will increase the difficulty or even violate the restrictions in the physical implementation of an LTG [3]. On the other hand, when using the summation of weights and threshold values as the cost function, the threshold logic network might be connected with more simpler functions, like AND or OR gates. This situation will increase the gate count and delay such that the compactness advantage of threshold logic is diminished.

Thus, to balance the objectives, in this paper, we adopt a hybrid cost function that integrates the both objectives together – considering the gate count, weights, and threshold values simultaneously in the proposed tool. For the objective of level of threshold logic circuits, it will be implicitly considered in this cost function. This is because using this cost function, the level of the threshold logic circuits will be moderate.

The proposed In&Out consists of Add-in and Remove-out techniques. The Add-in is to transform a threshold logic network into an implementation friendly network. This is because the final implementation will be linked to this netlist, no further technology mapping process is necessary. In the Add-in technique, we propose two theorems to efficiently determine if the structure transformation, i.e., adding an LTG, reduces the cost. In the Remove-out technique, we model the redundant gate removal as a SAT problem and use SAT-solvers to determine the redundancy in the network. Experimental results on a set of IWLS 2005 benchmarks show that the proposed tool can reduce the cost up to 10%.

The main contributions of this work are two-fold:

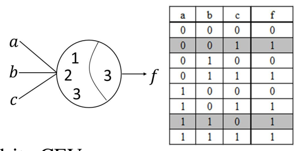
1. This is the first work that simultaneously considers the gate counts, the weights, and the threshold values of threshold logic network as the cost function during the synthesis. With this, the resultant threshold logic network is implementation friendly and practically cost-efficient.

2. We propose a synthesis tool, In&Out, that generates an implementation friendly threshold network.

2. Preliminaries

In this section, we review the characteristics of threshold logic, and introduce some background which will be used in our approach.

2.1. LTG Basics Each LTG can be represented as a *weight-threshold vector* $\langle w_1, w_2, \ldots, w_n; T\rangle$. For example, we can use $\langle 7, 5, 2, 2, 1, 1; 7\rangle$ to represent the LTG in Fig. 2(b). Additionally, the functionality of a threshold logic gate can be analyzed due to its output evaluation mechanism - the relationship between the summation of input-weights and the threshold value. According to this relationship, we can derive *don't care* bit in a positive-weight threshold logic gate. For example with Fig. 2, the input pattern 100000 produces the output of 1. Hence, we can derive that the patterns "1−−−−−" also have the output of 1, where "−" denotes don't care.



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

Fig. 3. An LTG and its CEVs.

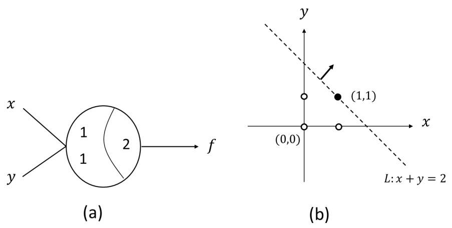


Fig. 4. (a) AND gate LTG. (b) Hyperplane and half-space of an AND gate.

2.2. Positive-negative weight transformation

According to the definition of an LTG, we know that the weights can be positive or negative numbers. However, the negative weights increase the difficulty for analyzing the threshold logic network. Therefore, we transform the negative weights and threshold values in the LTGs by applying the positive-negative weight transformation method [24].

This transformation method is as follows [24]. First, the negative weight is negated to the positive one and the inverter is passed to its corresponding input. Then, the threshold value of the LTG is modified by adding the absolute value of the negative weight. This transformation is repeated until all the weights become positive.

Since the transformation method is reversible, we will revert the transformation after synthesis for cost calculation and comparison.

2.3. Critical-effect vectors

An LTG has a *critical-effect* if and only if there exists an input assignment such that the output changes from 1 to 0 when any one of its inputs in this assignment changes from 1 to 0 [23]. The input vector that satisfies the mentioned property is called the *Critical-Effect Vector (CEV)* of the LTG. For instance in Fig. 3, an LTG $\langle 1, 2, 3; 3\rangle$ has two CEVs, $(a, b, c) = (0, 0, 1)$ and $(1, 1, 0)$. That is, if we change one input from 1 to 0 in the CEV, the output will also change from 1 to 0. CEVs can be derived by the algorithm proposed in [23].

2.4. Hyperplane and half-space

Hyperplane and half-space play an important role in the field of geometry and algebra. In this subsection, we focus on the relationship among the hyperplane, half-space and threshold logic gate.

The function of an LTG can be presented as a hyperplane in an n-dimensional space, i.e., $H : \sum_{i=1}^{n} x_i w_i = T$. The half-space described by $\sum_{i=1}^{n} x_i w_i \geq T$ and $\sum_{i=1}^{n} x_i w_i < T$ are represented as the positive half-space H^+ and the negative half-space H^-, respectively. When any point located in H^+ is applied to the threshold logic gate, the output is 1. Otherwise, the output is 0.

For example, the threshold logic gate $\langle 1, 1; 2\rangle$ as shown in Fig. 4(a) produces the output of 1 if and only if $x \times 1 + y \times 1 \geq 2$. The function of this threshold logic gate can be illustrated in a two-dimensional plane as shown in Fig. 4(b). In Fig. 4(b), when any point that is on or above the hyperplane $L : x + y = 2$ is applied to the threshold logic gate, the output f is 1. Therefore, we can represent the relationship among the hyperplane, half-space and threshold logic gate on an Euclidean space.

TABLE I. THE COST WITH DIFFERENT PARAMETER α.

α	cost	
	Fig. 2(a)	Fig. 2(b)
0	4	**1**
0.1	5.1	**3.4**
0.5	**9.5**	13
0.9	**13.9**	22.6
1.0	**15**	25

3. Add-in Technique

Before getting into the proposed synthesis method, we first introduce the proposed hybrid cost function as EQ(2)

$$cost = \alpha \cdot \sum_i (\sum_j w_{ij} + T_i) + (1-\alpha) \cdot |gate| \qquad (2)$$

where α is balance parameter about the summation of weights and threshold value, and the gate count, w_{ij} is the weight of input x_j in the gate i, T_i is the threshold value of gate i, and $|gate|$ is the gate count in the whole threshold network. EQ(2) is written as the linear function with respect to α parameter, and α is determined by designers based on the physical implementation of the used LTG device. However, this equation can be even expressed as quadratic with respect to α for a more sophisticated modeling. Nevertheless, in this work, for the demonstrative purpose of using this hybrid cost function, we adopt EQ(2) and set α as 0.5 to represent the equal importance of these two objectives.

Table I summarizes the cost values of threshold networks in Fig. 2 with different α settings in EQ(2). From Table I, we can realize that $\alpha = 0$ or $\alpha = 1$ is extremely biased and should be avoided.

Next, we introduce the proposed Add-in technique, which aims to obtain a transformed network with a lower cost. The Add-in technique first identifies an LTG, which is the gate to be transformed. We divide the transformation scenarios into two cases based on the characteristic of an LTG:

Case 1: For an LTG having a weight that is greater than or equal to the threshold value: In this scenario, the LTG produces the output of 1 without considering the values of other inputs if the input of this weight is 1. In other words, this input determines the output of 1 independently. With this property, we can split this input from the LTG. Thus, we connect this input and the remaining threshold gate with an OR gate (Add-in) at its transitive fanout cone. If such inputs are not unique in an LTG, we transform them simultaneously as Fig. 5 shows. Note that the overall functionality of the circuit after the transformation is not changed [21]. Here, we propose Theorem 1 with respect to this case as shown in Fig. 5 to determine the condition that the cost will be reduced after the transformation.

Theorem 1. *Given an n-input* $(x_1 \sim x_n)$ *LTG with k symmetric inputs* $x_1 \sim x_k$, $k \geq 1$, $f = \langle w_1, w_2, \ldots, w_k, w_{k+1}, \ldots, w_n; T \rangle, w_1 = w_2 = \ldots = w_k = T$*, the balance parameter* α *in the cost function of EQ(2), the cost is reduced after the transformation if and only if* $\alpha(kT - k - 1) > 1$.

Proof: $(\Rightarrow)$ Given an n-input LTG, $f = \langle w_1, w_2, \ldots, w_k, w_{k+1}, \ldots, w_n; T \rangle$, as shown in Fig. 5(a). The resultant network is as shown in Fig. 5(b) after the transformation. We would like to prove that the cost of circuit in Fig. 5(b) is less than that in Fig. 5(a). According to the cost function, we have an inequality

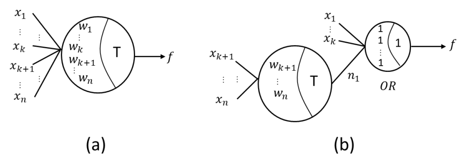


Fig. 5. The transformation of Case 1: (a) An original LTG. (b) The transformed LTGs.

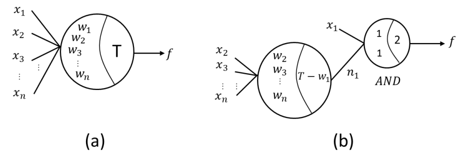


Fig. 6. The transformation of Case 2: (a) An original LTG. (b) The transformed LTGs.

$\alpha \cdot (w_1 + w_2 + \ldots + w_k + w_{k+1} + \ldots + w_n + T) + (1-\alpha) \cdot 1 > \alpha \cdot (w_{k+1} + \ldots + w_n + T + k + 1 + 1) + (1-\alpha) \cdot 2$. Since $w_1 = w_2 = \ldots = w_k = T$, $\alpha \cdot (T \times k + w_{k+1} + \ldots + w_n + T) + (1-\alpha) \cdot 1 > \alpha \cdot (w_{k+1} + \ldots + w_n + T + k + 2) + (1-\alpha) \cdot 2$. As a result, this inequality, $\alpha kT + (1-\alpha) > \alpha k + 2$, or $\alpha(kT - k - 1) > 1$ holds.

$(\Leftarrow)$ We can prove this condition by reversing the proof in $(\Rightarrow)$. ■

Before we introduce Case 2, we define two terms first.

Definition 1: An LTG is *useless* if and only if every input pattern produces the output of 0.

Definition 2: The input in an LTG is a *critical input* if and only if this LTG becomes useless after removing it.

Case 2: For an LTG having a critical input: For a critical input in an LTG, we can split it from the LTG and connect the remaining threshold gate with an AND gate (Add-in) at its transitive fanout cone as shown in Fig. 6. To preserve the overall functionality in the transformed network, we have to reduce the threshold value of this LTG by the weight of this critical input after the transformation [21]. Here, we propose Theorem 2 with respect to this case to determine the condition that the cost will be reduced after the transformation.

Theorem 2. *Given an n-input* $(x_1 \sim x_n)$ *LTG with the critical input* x_1*:* $f = \langle w_1, w_2, \ldots, w_n; T \rangle$*, the balance parameter* α *in the cost function of EQ(2), the cost is reduced in the transformed threshold logic network if and only if* $\alpha(2w_1 - 3) > 1$.

Proof: $(\Rightarrow)$ Given an n-input LTG, $f = \langle w_1, w_2, \ldots, w_n; T \rangle$ as shown in Fig. 6(a), the resultant network is as shown in Fig. 6(b) after the transformation. We would like to prove that the cost of network in Fig. 6(b) is less than that in Fig. 6(a). Therefore, we have an inequality $\alpha \cdot (w_1 + w_2 + w_3 + \ldots + w_n + T) + (1-\alpha) \cdot 1 > \alpha \cdot (w_2 + w_3 + \ldots + w_n + T - w_1 + 1 + 1 + 2) + (1-\alpha) \cdot 2$. As a result, this inequality, $2\alpha w_1 - 3\alpha > 1$ or $\alpha(2w_1 - 3) > 1$ holds.

$(\Leftarrow)$ We can prove this condition by reversing the proof in $(\Rightarrow)$. ■

4. Remove-Out Technique

In this section, we introduce the proposed Remove-out technique, which can remove redundant gates in a threshold network.

4.1. An example

As mentioned above, the function of an LTG can be represented as a hyperplane and half-space. Therefore, we can use a hyperplane and half-space to express an LTG. For

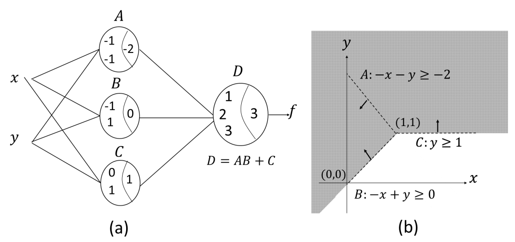


Fig. 7. (a) The threshold logic network. (b) The relationship of hyperplanes and half-spaces.

example in Fig. 4, we can derive the inequality $x + y \geq 2$ from the LTG $\langle 1, 1; 2\rangle$ based on the definition of an LTG. The equation $x+y = 2$ is the hyperplane and the positive half-space represents the on-set space as shown in Fig. 4(b). Since we can obtain the relationship of hyperplanes and half-spaces of LTGs on the Euclidean space, we can determine if redundant gates exist or not. For example, a threshold logic network is shown in Fig. 7(a) where the weights can be positive or negative. The hyperplanes and positive half-spaces of A, B, and C are shown in Fig. 7(b). Since the functionality of gate D is $AB + C$, we can obtain the resultant on-set space at f as shown in gray in Fig. 7(b) by intersection and union operations. However, we observe that this space can be generated by $B + C$ only. Thus, the gate A is redundant and can be removed.

4.2. Function derivation from the CEVs

Given an LTG, how to derive its function is important in the Remove-out technique. Here, we propose a theorem that helps derive the function of an LTG from its CEVs.

Theorem 3. *Given an n-input $(x_1 \sim x_n)$ LTG G with a set of its CEVs. The function of G can be derived by ORing the P_i of each CEV V_i, where P_i is the product of input variables that are assigned 1 in the CEV V_i.*

Proof: Given an n-input $(x_1 \sim x_n)$ LTG G, and assume that it has m CEVs, $V_1, \ldots, V_i, \ldots, V_m$. From the definition of a CEV, the output of V_i is 1. Since all the inputs changing from 0 to 1 in a V_i will cause the output unchanged, i.e., output is still 1, these inputs are don't care bits of V_i. Thus, the function of V_i is P_i where P_i is the product of inputs that are assigned 1. Furthermore, the function of an LTG G can be determined by its CEVs. Thus, the function of $G = \sum_{i=1}^{m} P_i$. ■

For example in Fig. 3, the inputs of the LTG are a, b, c. The CEVs of f is $\{V_1 = 001, V_2 = 110\}$. Hence, the function of f is derived as $\sum_{i=1}^{2} P_i = P_1 + P_2 = c + ab$.

4.3. SAT-based identification of redundant gates

In the example of Fig. 7, we identify the redundancy of gate A from the geometry perspective. However, for the gate with more inputs, e.g., 6, its hyperplane is within a 6-dimensional space. For this case, it is hard to have a geometric understanding about the redundancy. Thus, alternatively, we model this redundancy identification as a SAT problem and use SAT-solvers to determine if a gate is redundant. Here, to tradeoff the effectiveness and the efficiency of redundancy identification, we only identify the redundancies within a two-level sub-circuit. Fig. 7(a) is an example of two-level sub-circuit. In Fig. 7(a), the rear gate D is called the *functional gate* and its fanin gates A, B, and C are called the *candidate gates*.

Next, we explain the SAT modeling of this Remove-out technique. Since the function in the functional gate is in *Sum-Of-Product (SOP)* form by Theorem 3, we would like to examine if there exists any candidate gate contained by other candidate gates through AND, OR operations. Thus, the modeling contains two phases: AND-phase and OR-phase. The AND-phase is to examine the relationship between internal variables, while the OR-phase is to examine the relationship between product terms.

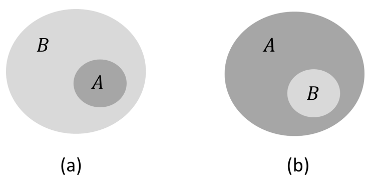


Fig. 8. The relationship between on-sets. (a) A is completely contained by B. (b) B is completely contained by A.

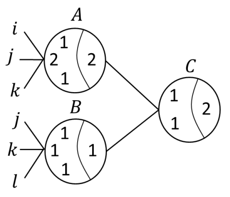


Fig. 9. An example of AND-phase.

1) AND-phase: In this phase, we focus on the relationship between internal variables in a product term. For two internal variables in a product term, if the on-set space of a variable (gate) A is completely contained by that of another variable (gate) B, as shown in Fig. 8(a), the variable (gate) B is redundant. Thus, we can model this situation as EQ(3)[1]

$$A \cdot \bar{B} = 1 \tag{3}$$

such that when the SAT-solver returns UNSAT for EQ(3), B is redundant. To simultaneously consider that either variable A or B in a product term is redundant, we update the model as shown in EQ(4).

$$A \cdot \bar{B} \oplus \bar{A} \cdot B = 1 \tag{4}$$

If EQ(4) is SAT, that means $\{A \cdot \bar{B} = 1, \bar{A} \cdot B = 0\}$ or $\{A \cdot \bar{B} = 0, \bar{A} \cdot B = 1\}$. By considering $\bar{A} \cdot B = 0$ or $A \cdot \bar{B} = 0$ we known that either A or B is redundant. However, EQ(4) is modeled as SAT for identifying the redundancy, which is improper for SAT-solving process. Thus, we add a complement on it as shown in EQ(5),

$$\overline{A \cdot \bar{B} \oplus \bar{A} \cdot B} = 1 \tag{5}$$

such that when EQ(5) is UNSAT, either A or B is redundant. After this, we then further check the equations like EQ(3) to determine which variable is redundant.

For example, as shown in Fig. 9, gates A and B are the candidate gates and gate C is the functional gate. The function of gate A, gate B, and gate C are $A = ik + j$, $B = j + k + l$ and $C = AB$, respectively. Thus, we check the equation $\overline{(ik + j) \cdot (\overline{j + k + l}) \oplus (\overline{ik + j}) \cdot (j + k + l)} = 1$ according to EQ(5). The result is UNSAT, which indicates either A or B is redundant. Then we further check the function $(ik + j) \cdot (\overline{j + k + l}) = 1$, and find that it is UNSAT. As a result, the candidate gate B is redundant.

Similarly, for the product term with more than two variables, e.g., A, B, and C, EQ(5) can be extended as EQ(6). We apply the similar procedure to determine the redundancy.

$$\overline{A \cdot \bar{B} \oplus \bar{A} \cdot B} \cdot \overline{A \cdot \bar{C} \oplus \bar{A} \cdot C} \cdot \overline{B \cdot \bar{C} \oplus \bar{B} \cdot C} = 1 \tag{6}$$

[1]If the relationship of A and B is like Fig. 8(b), the modeling is expressed as $\bar{A} \cdot B = 1$.

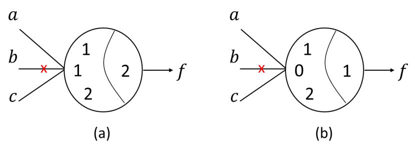


Fig. 10. Update the weights and threshold values in the AND-phase. (a) The original gate. (b) The updated gate.

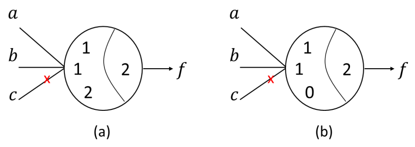


Fig. 11. Update the weights and threshold values in the OR-phase. (a) The original gate. (b) The updated gate.

2) OR-phase: In the OR-phase, we focus on the relationship between product terms in a functional gate. For example, in Fig. 7(a), we examine the relationship between two product terms AB and C. Since this is an OR operation between two product terms, the product term with a smaller on-set space is redundant when there exists another product term that completely contains it.

We also use the modeling in EQ(3) to find the redundant product term where A and B represent product terms instead of variables. Note that in the OR-phase, when EQ(3) is UNSAT, it means that the product term A rather than B is redundant.

4.4. Weights and threshold values update

After removing out the redundant gates, we have to update the weights and threshold value of the functional gate accordingly. The functionality of the functional gate after the update is still the same.

3) AND-phase: For the redundant inputs, its weights are changed to 0. Then, the threshold value is reduced by the weights of redundant inputs. For example, as shown in Fig. 10(a), assume that b is identified as redundant in the AND-phase. Thus, we set 0 to its weight and update the threshold value by reducing the weight of input b, as shown in Fig. 10(b)[2].

4) OR-phase: For the redundant product term in the OR-phase, we change its weights to 0 and make the threshold value intact. For example, as shown in Fig. 11(a), assume that c is identified as redundant in the OR-phase. The weight of this redundant product term is changed to 0, as shown in Fig. 11(b)[2].

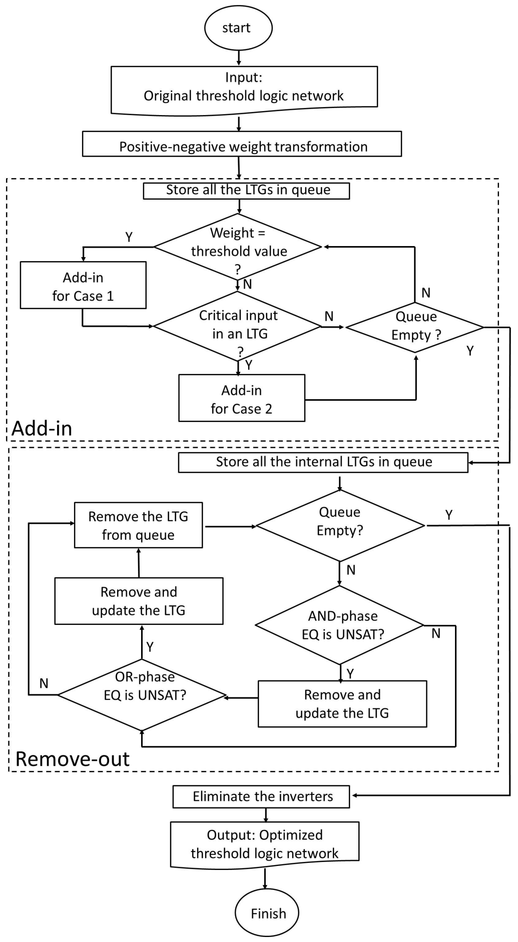
5. Overall algorithm

Fig. 12 shows the overall algorithm of our approach. The input is a threshold logic network and the output is an optimized threshold logic network. The positive-negative weight transformation is the preprocessing stage. The next stage is to apply the Add-in technique such that the network is restructured for reducing the cost while preserving the overall functionality. Next, we conduct the Remove-out technique for removing the redundant gates. At the end of our algorithm, we perform the positive-negative weight transformation again to eliminate the inverter gates.

6. Experimental results

We implemented the proposed tool in C++ language. The experiments were conducted on a 3.0 GHz Linux platform

[2]For the inputs with the weights of 0, we can also directly remove them from the functional gate.



start
Input: Original threshold logic network
Positive-negative weight transformation
Store all the LTGs in queue
Weight = threshold value ?
Add-in for Case 1
Critical input in an LTG ?
Queue Empty ?
Add-in for Case 2
Add-in
Store all the internal LTGs in queue
Remove the LTG from queue
Queue Empty?
Remove and update the LTG
AND-phase EQ is UNSAT?
OR-phase EQ is UNSAT?
Remove and update the LTG
Remove-out
Eliminate the inverters
Output: Optimized threshold logic network
Finish
Y
N

Fig. 12. The overall flow of our tool.

with Intel Xeon E5530. The benchmarks were selected from IWLS 2005 [36] and MCNC. These benchmarks were initially synthesized as threshold logic network, using the number of primary input as the fanin number constraint.

In the experiments, we demonstrate the capability of the proposed tool in the optimization of threshold network in terms of the cost reduction. Table II summarizes the experimental results. Column 1 lists the benchmarks. Columns 2 and 3 show the number of LTGs and the summation of weights and threshold values of the original benchmarks, respectively. Column 4 lists the original cost where the balance parameter α is set as 0.5 for the ease of demonstration. Columns 5 and 6 are the number of instances in Case 1 and Case 2, respectively. Columns 7 and 8 list the number of redundant gates which have a single fanout in the AND-phase and OR-phase, respectively. Columns 9 to 11 show the corresponding results after applying our tool. Column 12 shows the ratio of cost reduction. Finally, the CPU time measured in second is listed in the last column.

According to Table II, the ratio of cost reduction after the optimization can be up to 10%. However, some benchmark like *C1355* is only a little. Therefore, the amount of cost reduction strongly depends on the initial circuit structure. This is also the limitation of this optimization approach. In fact, obtaining a global optimal result for a benchmark is intractable. Note that here we do not compare this work with the previous works. This is because we cannot obtain the optimized networks from the previous works. Also, the cost functions they used are different from ours such that the comparison might be unjustified. Furthermore, if we set $\alpha = 0$ in EQ(2) to compare the gate count or set $\alpha = 1$ to compare the summation of

TABLE II. THE EXPERIMENTAL RESULTS OF OUR TOOL.

				Ours								
	Initial			Add-in		Remove-out		Optimized				
Benchmark	|LTG|	Sum	Cost	|Case 1|	|Case 2|	AND-phase	OR-phase	|LTG|	Sum	Cost	Impr.(%)	T(s)
C1908	287	1316	801.5	1	8	0	24	271	1202	736.5	8.11	7.46
usb_phy	288	1362	825	4	13	2	3	305	1310	807.5	5.45	14.82
t481	320	1613	966.5	18	12	0	4	334	1481	907.5	6.1	4.84
C1355	340	1662	1001	2	0	0	0	344	1650	992	0.9	12.22
rot	354	1606	980	3	13	1	9	340	1462	901	8.06	6.94
alu4	372	1768	1070	4	7	0	4	377	1703	1040	2.8	11.65
x3	390	1749	1069.5	0	6	2	3	378	1692	1035	3.23	4.02
i2c	503	2928	1715.5	23	25	0	1	547	2553	1550	9.65	5.09
frg2	509	3021	1765	64	69	0	16	606	2618	1612	8.67	4.64
pci_spoci_ctrl	566	2718	1642	21	25	2	6	573	2526	1549.5	5.63	5.55
simple_spi	570	2385	1477.5	7	17	0	2	581	2291	1436	2.81	6.01
pair	712	3330	2021	9	25	4	9	699	3137	1918	5.1	29.69
dalu	782	3135	1958.5	6	3	3	4	754	3022	1888	3.73	1619.14
C5315	1076	3925	2500.5	0	1	0	8	1049	3860	2454.5	1.84	16.12
s9234	1080	5791	3435.5	17	60	5	20	1142	5044	3093	9.78	1187.58
C7552	1520	5362	3441	1	1	1	14	1483	5241	3362	2.31	1918.29
C6288	1818	7619	4718.5	12	11	0	0	1829	7361	4495	4.74	46.14
s13207	1935	7689	4812	17	51	2	26	1837	6743	4290	10.85	368.95
systemcdes	2070	9096	5583	7	46	13	11	2016	8595	5305.5	4.97	46.01

weights and threshold values against the previous works, that does not make any sense due to improper and biased cost function.

7. Conclusion

In this paper, we propose a tool, In&Out for threshold logic network optimization. The cost function of the optimization is also generalized by considering the factors from the physical implementation perspective. The Add-in technique reduces the cost by decreasing the weights and threshold value while the Remove-out technique removes the redundant gates in the threshold logic network. The experimental results demonstrate the improvements after the optimization.

REFERENCES

[1] M. J. Avedillo *et al.*, “Multi-Threshold Threshold Logic Circuit Design using Resonant Tunnelling Devices,” *Electron. Lett.*, 2003, pp. 1502-1504, vol. 39.

[2] I.A. Basheer *et al.*, “Artificial neural networks: fundamentals, computing, design, and application,” *Journal of Microbiological Methods*, 2000, pp. 3-31.

[3] V. Beiu *et al.*, “VLSI Implementations of Threshold Logic-A Comprehensive Survey,” *IEEE Transactions on Neural Networks*, 2003, pp. 1217-1243, vol. 14.

[4] V. Beiu, “On Existential and Constructive Neural Complexity Results,” *Neural Networks and Computational Intelligence*, 2003, pp. 63-72.

[5] P. Celinski *et al.*, “State-of-the-Art in CMOS Threshold-Logic VLSI Gate Implementation,” *VLSI Circuits and Systems Conference*, 2003, pp. 53-64.

[6] Y.-C. Chen *et al.*, “Automated Mapping for Reconfigurable Single-Electron Transistor arrays,” *Proc. DAC*, 2011, pp. 878-883.

[7] Y.-C. Chen *et al.*, “A Synthesis Algorithm for Reconfigurable Single-Electron Transistor Arrays,” *ACM Journal on Emerging Technologies in Computing System*, 2013, p. Article 5, vol. 9.

[8] Y.-C. Chen *et al.*, “Fast Synthesis of Threshold Logic Networks with Optimization,” *Proc. ASP-DAC*, 2016, pp. 486-491.

[9] C.-E. Chiang *et al.*, “On Reconfigurable Single-Electron Transistor Arrays Synthesis using Reordering Techniques,” *Proc. DATE*, 2013, pp. 1807-1812.

[10] F. Deliang *et al.*, “Design and Synthesis of Ultralow Energy Spin-Memristor Threshold Logic,” *IEEE Trans. Nanotechnology*, 2014, 13(3):574-83.

[11] S. Draghici *et al.*, “A VLSI-optimal constructive algorithm for classification problems,” *International Journal of Smart Engineering System Design*, 1997, pp. 145-151.

[12] S. Eachempati *et al.*, “Reconfigurable Bdd-based Quantum Circuits,” *Proc. Int. Symp. on Nanoscale Architectures*, 2008, pp. 61-67.

[13] P. Gupta *et al.*, “Automatic Test Generation for Combinational Threshold Logic Networks,” *IEEE Trans. CAD*, 2008, pp. 1035-1045, vol. 16.

[14] T. Gowda *et al.*, “Identification of Threshold Functions and Synthesis of Threshold Networks,” *IEEE Trans. CAD*, 2011, pp. 665-677, vol. 30.

[15] T. Gowda *et al.*, “A Non-ILP Based Threshold Logic Synthesis Methodology,” *Proc. International Workshop on Logic and Synthesis*, 2007, pp. 222-229.

[16] T. Gowda *et al.*, “Decomposition Based Approach for Synthesis of Multi-Level Threshold Logic Circuits,” *Proc. Asia and South Pacific Design Automation Conf.*, 2008, pp. 125-130.

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

[18] D. Hampel *et al.*, “Threshold logic,” *IEEE Spectrum*, 1971, pp. 32-39, vol. 8.

[19] S. L. Hurst, “Sequential Circuits using Threshold Logic Gates,” *Int. Journal of Electronics*, 1970, pp. 495-499, vol. 29.

[20] Y. Jin *et al.*, “Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies,” *IEEE Trans. on Systems*, 2008, pp. 397-415.

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

[22] C. Lageweg *et al.*, “A Linear Threshold Gate Implementation in Single Electron Technology,” *Proc. Comput. Soc. Workshop VLSI*, 2001, pp. 93-98.

[23] C.-C. Lin *et al.*, “Rewiring for Threshold Logic Circuit Minimization,” *Proc. DATE*, 2014, pp. 1-6.

[24] S. Muroga, “Threshold Logic and its Applications,” 1971, New York, NY: John Wiley.

[25] A. Neutzling *et al.*, “Threshold Logic Synthesis Based on Cut Pruning,” *Proc. ICCAD*, 2015, pp. 494-499.

[26] C. Pacha *et al.*, “Resonant Tunneling Device Logic Circuit,” 1999, DortmundGerhard-Mercator University of Duisburg, Germany, Tech. Rep.

[27] M. Perkowski *et al.*, “Logic Synthesis for Regular Fabric Realized in Quantum dot Cellular Automata,” *Journal of Multiple-Valued Logic and Soft Comput.*, 2004, pp. 768-773.

[28] V. Saripalli *et al.*, “Energy-delay Performance of Nanoscale Transistors Exhibiting Single Electron Behavior and Associated Logic Circuits,” *Journal of Low Power Electronics*, 2010, pp. 415-428,vol. 6.

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

[30] P. Venkataramani *et al.*, “Sequential Circuit Design in Quantumdot Cellular Automata,” *Proc. Nanotechnology Conf.*, 2008, pp. 534-537.

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

[32] R. O. Winder, “Threshold Logic,” 1962, Ph.D. dissertation, Princeton University, Princeton, NJ.

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

[34] R. Zhang *et al.*, “Threshold Network Synthesis and Optimization and its Application to Nanotechnologies,” *IEEE Trans. Comput-Aided Design Integr. Circuits Syst.*, 2005, 24(1):107-18.

[35] Y. Zheng *et al.*, “SAT-based Equivalence Checking of Threshold Logic Designs for Nanotechnologies,” *Proc. Great Lake Symp. VLSI*, 2008, pp. 225-230.

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