Area-Oriented Threshold Logic Circuit Synthesis Using Negative Weights

Yu-Chuan Yen[1], Fu-Cheng Cai[1], Yi-Ting Li[1], Wuqian Tang[1], Yung-Chih Chen[2], Ihao Chen[3], and Chun-Yao Wang[1]

[1]National Tsing Hua University
[2]National Taiwan University of Science and Technology
[3]Incentia Design Systems Inc.

2026 27th International Symposium on Quality Electronic Design (ISQED) | 979-8-3315-8361-3/26/$31.00 ©2026 IEEE | DOI: 10.1109/ISQED69900.2026.11534711

***Abstract*—This paper presents a novel approach to Threshold Logic Network (TLN) synthesis, which effectively leverages negative weights to minimize area costs. Traditional synthesis methods predominantly use positive weights, which limits their potential in certain hardware implementations. Our approach reduces the area cost of threshold logic gates without sacrificing circuit expressiveness. Using the positive-negative weight transformation method, we achieve significant area reductions while maintaining the expressive capabilities of the TLN. Experimental results demonstrate our approach's consistent benefits, achieving reduction ratio up to 11.97% in modern threshold logic circuit design, thus showcasing its effectiveness.**

I Introduction

A Threshold Function (TF) is a function that can be expressed using a single threshold logic gate (TLG). Each input x_i in a TLG has an associated weight w_i. Furthermore, a TF is characterized by a threshold value T. Represented as $[w_1, w_2, ..., w_n; T]$, a TF returns 1 if the summation of the weights of active inputs (i.e., $x_i = 1$) is greater than or equal to T; otherwise, it returns 0. The behavior of an n-input TLG, which implements the TF, can be mathematically expressed as:

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

These gates, also known as 1st-order TLGs (1-TLGs), offer an alternative to conventional Boolean networks, potentially implementing complex functionalities with a single gate. For example, the threshold function $[1, 1, 1; 2]$, which can be implemented with a single 1-TLG, is equivalent to the Boolean function $x_1x_2 + x_2x_3 + x_1x_3$, which requires three AND gates and one OR gate in a traditional Boolean network implementation. To further enhance expressiveness, 2nd-order TLGs (2-TLGs) [25] introduce additional weights activated by the product of any two inputs, allowing the realization of even more complex functions. However, this enhanced expressiveness comes at the cost of increased complexity in synthesis and implementation.

This work was supported in part by the National Science and Technology Council (Taiwan) under Grant NSTC 112-2218-E-007-014, Grant NSTC 112-2221-E-007-106-MY2, Grant NSTC 112-2221-E-007-108, Grant NSTC 112-2425-H-007-002, Grant NSTC 113-2425-H-007-004, Grant NSTC 113-2640-E-011-003, Grant NSTC 113-2221-E-007-082-MY3, Grant NSTC 114-2221-E-007-125-MY3, Grant NSTC 114-2218-E-007-002, Grant NSTC 114-2221-E-007-126-MY3, Grant NSTC 114-2425-H-007-002, and National Tsing Hua University under NTHU 111A0131EX and 112A0241EX.

Threshold logic related research is flourishing recently [10, 11, 15, 21, 23]. Threshold Logic Networks (TLNs), which consist of TLGs, are designed to represent complex Boolean functions through a networked arrangement of threshold gates. TLN synthesis (TLS) [4, 7, 9, 12, 13, 14, 17, 24] is an area of significant research interest, with efforts focused on optimizing metrics such as gate count, circuit depth, or circuit area. Emerging hardware technologies such as memristor[5, 18], Resonant Tunneling Diodes (RTDs) [1, 2, 19, 20], Single-Electron Transistors (SETs) [6], and Quantum Cellular Automata (QCAs) [22] offer distinct advantages for threshold logic implementation. The choice of hardware implementation for TLG greatly influences the area cost [3]. For instance, implementations using SETs and QCAs often relate the area cost to the number of gate fan-in. RTDs, however, present a different area cost model, where the area is proportional to the absolute summation of weights, meaning both positive and negative weights have an equal impact to the cost of implementation.

Among the various hardware implementation options, RTDs stand out due to their potential for high-speed, compact design, room-temperature operation, and the ability to express more complex functionality within a single gate [3]. However, many existing TLS algorithms primarily focus on positive weights, thus not fully exploiting the balanced values of positive and negative weights in RTD-based implementations. Hence, this paper aims to effectively utilize negative weights by proposing a synthesis approach specifically for RTD-based implementations of TLNs.

The main contributions of this paper are threefold.

1) We propose the first TLN synthesis approach that leverages the properties of negative weights under RTD-based implementations.
2) By employing negative weights, we aim to reduce the area cost of TLG without compromising the expressiveness of the TLNs.
3) Our approach is applicable to both 1-TLG and 2-TLG structures, offering a versatile solution for threshold logic designs.

II Preliminaries

A. 1st-order threshold logic gate

An n-input 1-TLG comprises n weights $(w_1, \ldots, w_n)$ and a threshold value T. In this work, we denote positive weights

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

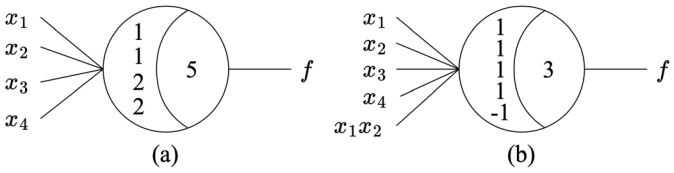

Figure 1: Examples of TLGs. (a) 1-TLG $[1, 1, 2, 2; 5]$. (b) 2-TLG $[1, 1, 1, 1, -1, 0, 0, 0, 0, 0; 3]$.

as w_i^+ and negative weights as w_i^-, with w_i representing the general integer weights that may be either positive or negative.

The output of a 1-TLG is 1 if the weighted summation of the input pattern is greater than or equal to T; otherwise, the output is 0. A 1-TLG can be represented as a weight-threshold vector $[w_1, w_2, \ldots, w_n; T]$[25].

For instance, consider a 4-input 1-TLG $[1, 1, 2, 2; 5]$. With an input pattern of $x_1x_2x_3x_4 = 0111$, the last three weights are activated, and the weighted summation equals the threshold value $(0\times1+1\times1+1\times2+1\times2 = 5)$, resulting in an output of 1. However, for an input pattern of $x_1x_2x_3x_4 = 1110$, the weighted summation is only 4, which is less than the threshold value, thus producing an output of 0. Figure 1(a) shows the graphical representation of a 1-TLG, and its Boolean function is $f = (x_1 + x_2)x_3x_4$.

B. 2^{nd}-order threshold logic gate

An n-input 2-TLG has $n + \binom{n}{2}$ weights. In addition to the n weights associated with single inputs, there are $\binom{n}{2}$ weights associated with products of any two inputs, termed 2^{nd}-order weights (2-weights). Similar to 1-TLGs, the output is 1 if the summation of the activated weights is greater than or equal to the threshold value; otherwise, the output is 0. An n-input 2-TLG is represented as $[w_1, \ldots, w_n, w_{1,2}, w_{1,3}, \ldots, w_{n-1,n}; T]$. In a 2-TLG, the majority of the 2-weights are usually 0. Hence, for clarity, we only present non-zero weights and their corresponding inputs when illustrating a 2-TLG as shown in Figure 1(b).

For example, the function $f = (x_1 + x_2)x_3x_4$ has a 1-TLG form as shown in Figure 1(a). As in 2-TLG, x_1, x_2, and x_1x_2 can form an OR operation using weights $1, 1$, and -1 (the weighted summation is 1 only when $x_1 + x_2 = 1$). Therefore, we can represent $x_o = (x_1+x_2)$ using the 2-TLG $[1, 1, -1; 1]$. If we use x_o to represent $(x_1 + x_2)$, the expression $f = (x_1 + x_2)x_3x_4$ can be replaced by a 3-input AND operation $x_ox_3x_4$, where its 1-TLG representation is $[1, 1, 1; 3]$. By substituting x_o with the 2-TLG $[1, 1, -1; 1]$, the resultant 2-TLG for f is $[1, 1, 1, 1, -1, 0, 0, 0, 0, 0; 3]$, where only $w_{1,2}$ is a non-zero 2-weight. This is an example illustrating how to construct the corresponding 2-TLG of a function. Figure 1(b) graphically depicts this 2-TLG, which represents the Boolean function $f = (x_1 + x_2)x_3x_4$ as well.

C. RTD-based threshold logic implementation

Resonant tunneling diodes (RTDs) serve as advantageous hardware implementations for TLGs, and their current-voltage characteristics is shown in Figure 2(a). The RTD-based circuit is primarily based on the Monostable-Bistable Logic Element (MOBILE) devices. The MOBILE, as Figure 2(a) shows, is a rising edge triggered current-controlled gate comprising two series-connected RTDs driven by a switching bias voltage (V_{bias}). At low V_{bias}, both *Load* and *Driver* RTDs are on, and the circuit is monostable (stable at 0). As V_{bias} increases to a specific value, only the RTD with the lower peak current turns off, determining the output state. The output is high if the *Driver* RTD switches; and is low if the *Load* RTD switches. Assuming equal current densities for both RTDs, peak currents are proportional to RTD areas λ_L (*Load*) and λ_D (*Driver*), respectively. For example, with an appropriate V_{bias}, the behavior of V_{out} in Figure 2(a) can be expressed mathematically as EQ(1):

$$V_{out} = \begin{cases} 1, & \text{if } \lambda_L \geq \lambda_D \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

A TLG can be implemented by associating an input with each RTD using a heterojunction field-effect transistor (HFET) to control the TLG. For example, Figure 2(b) shows a 2-TLG that is defined as EQ(2):

$$V_{out} = \begin{cases} 1, & \text{if } x_1w_1^+ + x_2w_2^+ + x_1x_2w_{1,2}^- \geq T \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where $w_1^+, w_2^+ > 0$ and $w_{1,2}^- < 0$. The behavior of V_{out} in Figure 2(b) can be expressed as EQ(3):

$$V_{out} = \begin{cases} 1, & \text{if } x_1\lambda_1 + x_2\lambda_2 + \lambda_L \geq x_1x_2\lambda_3 + \lambda_D \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

With mathematical manipulations in EQ(2) and EQ(3), it can be shown that the RTD areas $(\lambda_1, \lambda_2, \lambda_3)$ determine the weights $(|w_1^+|, |w_2^+|, |w_{1,2}^-|)$, and the RTD area ($\lambda_D$ and λ_L) determines the threshold value T. Specifically, $|w_1^+| = \lambda_1,\ |w_2^+| = \lambda_2,\ |w_{1,2}^-| = \lambda_3,\ T = \lambda_D - \lambda_L$.

As discussed in the previous introduction to the behavior of MOBILE device, there is no difference in area cost between positive and negative weights in a TLG; the only distinction lies in the placement of the RTDs in *Load* or *Driver*. Therefore, the area cost model of 1-TLG and 2-TLG can be expressed as EQ(4).

$$\text{For a 1-TLG}: A_{1-TLG} = \sum_{i=1}^{n} |w_i| + |T|$$

$$\text{For a 2-TLG}: A_{2-TLG} = \sum_{i=1}^{n} |w_i| + (\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |w_{i,j}|) + |T| \quad (4)$$

D. Positive-Negative weights transformation

As mentioned, RTD-based threshold logic implementations intrinsically support the coexistence of positive and negative weights. This work proposes a novel synthesis approach that leverages the properties of negative weights, through the positive-negative weight transformation [16].

The transformation method for a negative (positive) weight in a TLG is described below. First, a negative (positive) weight is negated into a positive (negative) one, and we apply a NOT operation to the corresponding variable of this negative (positive) weight. The threshold value then is decreased by the negative (positive) weight.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

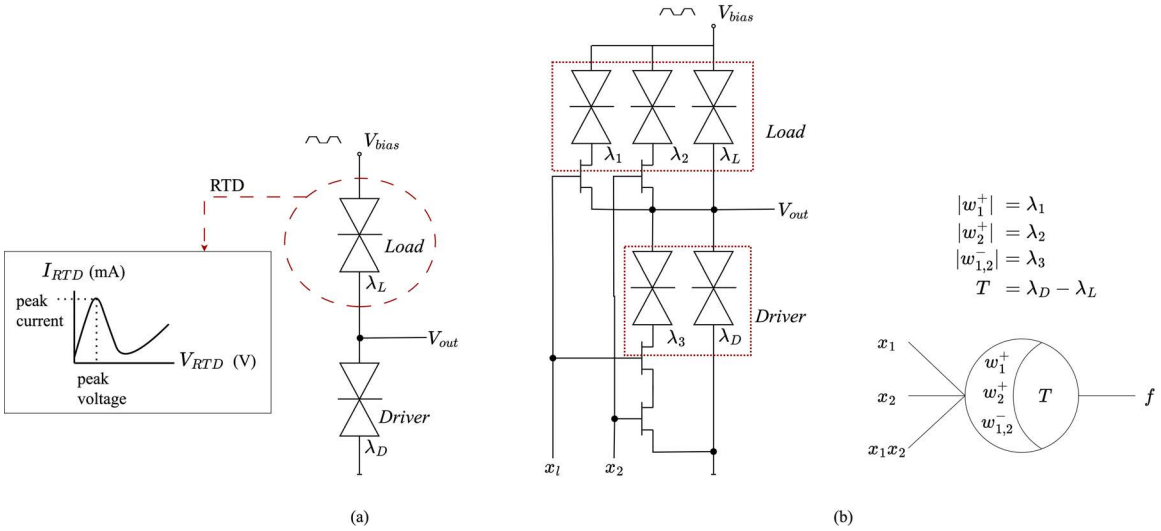

Figure 2: (a) A basic MOBILE consisting of two RTDs. (b) A MOBILE circuit for threshold logic and its equivalent TLG.

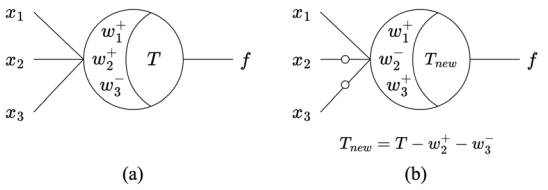

Figure 3: (a) A TLG before applying the transformation. (b) A TLG after applying the transformation.

Figure 3 illustrates a transformation example where positive weight w_2^+ becomes negative weight w_2^-, and negative weight w_3^- becomes positive weight w_3^+. First, we negate the weights from w_2^+ to w_2^- and from w_3^- to w_3^+. Then we apply NOT operations to variables x_2 and x_3. The new threshold value T_{new} is then decreased by w_2^+ and w_3^-.

E. Rewiring with critical wire

In the previous study of TLS [9], a critical wire rewiring is a technique for logic restructuring. A critical wire is defined as an input whose removal will cause the summation of the remaining weights to fall below the threshold value, thereby always forcing the output of the TLG to zero. Hence, it is imperative that the critical wire is set to 1 to observe the impacts from other input combinations.

The technique of critical wire rewiring involves three steps:

1) Identification of the critical wire: This is achieved by analyzing the weight contribution of each input and determining if removing an input would lead to the output of zero. If the summation of remaining weights, after excluding a certain wire, is less than the threshold value, that wire is classified as critical.
2) Setting the TLG propagation condition: An additional AND gate is introduced to propagate the TLG output only under the condition that the critical wire is set to 1.
3) Adjustment for the critical wire's impact: After rewiring the critical wire, the threshold value of the original TLG must be adjusted. This adjustment is necessary for removing the influence of the weight of the critical wire on the TLG.

For instance, for the 5-input TLG $[6, 3, 3, 2, 1; 10]$ in Figure 4(a), input x_1 is critical because the summation of the remaining weights is less than the threshold value ($3+3+2+1 < 10$). Therefore, an AND gate is added in the fanout of the TLG to ensure that the value is propagated only when x_1 is set to 1. Furthermore, since the propagated output is constrained to the condition of $x_1 = 1$, the original threshold value needs to be adjusted by subtracting the impact of w_1 ($10-6 = 4$). Figure 4(b) shows the resultant TLN after rewiring x_1.

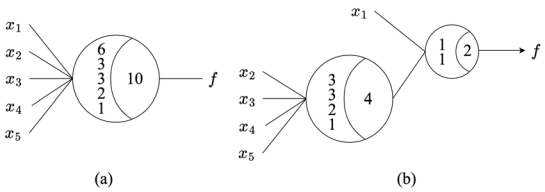

Figure 4: (a) The TLG before applying the critical wire rewiring. (b) The resultant TLN after applying the critical wire rewiring.

III Proposed Approach

A. Cost minimization using P-N weights transformation

This work presents a novel approach for reducing the area cost of a TLG through the transformation between positive and negative weights. As outlined in Section II-C, the area cost of a TLG can be represented as $A_{1-TLG} = \sum_{i=1}^{n} |w_i| + |T|$ or $A_{2-TLG} = \sum_{i=1}^{n} |w_i| + (\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |w_{i,j}|) + |T|$.

To analyze the transformation's effect, we will refer to the scenario in which all weights undergo transformation as *full transformation*. We will conduct the analysis using 1-TLG, while the application of 2-TLG will be discussed in Section

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

III-C. In this analysis, the signs of the weights are crucial, and thus we will rearrange the weights accordingly. For an n-input TLG with k positive weights and $n-k$ negative weights, we will reorder the weights such that the positive weights are in the first k places, while the remaining negative weights will be placed from $k+1$ to n. In other words, the TLG to be analyzed is: $[w_1^+, \ldots, w_k^+, w_{k+1}^-, \ldots, w_n^-; T]$. It is important to note that the superscript of each symbol $w_i^{+/-}$ indicates the sign status of the weight; therefore, after the full transformation, the originally positive weights $w_1^+, \ldots, w_k^+$ will be transformed into $w_1^-, \ldots, w_k^-$. The weights $w_{k+1}^-, \ldots, w_n^-$ will similarly be transformed into $w_{k+1}^+, \ldots, w_n^+$. Figure 5 illustrates an example of *full transformation* to an n-input TLG, while EQ(5) presents the area costs before and after the *full transformation*.

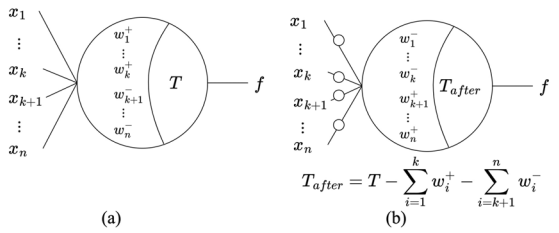

Figure 5: (a) A TLG before the *full transformation*. (b) A TLG after the *full transformation*.

$$
\begin{aligned}
A_{before} &= \sum_{i=1}^{n} |w_i| + |T| \\
&= \sum_{i=1}^{k} w_i^+ + \sum_{i=k+1}^{n} (-w_i^-) + |T| \\
A_{after} &= \sum_{i=1}^{k} (-w_i^-) + \sum_{i=k+1}^{n} (w_i^+) \\
&\quad + |T - \sum_{i=1}^{k} w_i^+ - \sum_{i=k+1}^{n} w_i^-| \\
&= \sum_{i=1}^{n} |w_i| + |T - \sum_{i=1}^{k} w_i^+ - \sum_{i=k+1}^{n} w_i^-| \\
&= \sum_{i=1}^{n} |w_i| + |T_{after}|
\end{aligned}
\tag{5}
$$

From EQ(5), it is clear that any transformation only affects the $|T_{after}|$ term in area cost. Therefore, identifying combinations of transformations that result in $|T_{after}| < |T|$ will further reduce the area cost of the TLG. Noted that each weight transformation introduces an additional NOT gate. Therefore, when considering the area cost, the cost of the NOT gate (A_{not}=1) must be taken into account. Algorithm 1 effectively identifies a subset of weights to be transformed, ensuring that the resultant TLG is optimal in terms of area cost.

In Algorithm 1, T_{before} and T_{after} denote the threshold values of the TLG before and after the application of the Threshold Value Optimization Strategy (TVOS), respectively. The weights $w_{(i,before)}$ and $w_{(i,after)}$ correspond to the i^{th} weight of the TLG prior to and following the TVOS. The algorithm initiates by setting a high initial best cost. For each candidate transformation set, it iteratively assesses the impact of weight transformations by subtracting the weights within the set from the current threshold. The cost is calculated based on the absolute value of the adjusted threshold, as well as the area cost associated with the introduced NOT gates, which corresponds to the number of weights being transformed. By comparing the calculated cost against the best cost found, Algorithm 1 effectively selects the optimal subset of weights for transformation, thereby minimizing the overall area cost.

Input: TLG_{before}
Output: TLG_{after}
$bestCost \leftarrow \infty$;
$bestTransformationSet \leftarrow [\ ]$;
$S \leftarrow$ all candidate sets;
for *set* s *in* S **do**
  $currentThreshold \leftarrow T_{before}$;
  foreach *index* i *in* s **do**
    $currentThreshold \leftarrow currentThreshold - w_{(i,before)}$;
  end
  $currentCost \leftarrow |currentThreshold| + s.size()$;
  if $currentCost < bestCost$ **then**
    $bestCost \leftarrow currentCost$;
    $bestTransformationSet \leftarrow s$;
  end
end
foreach *index* i *in* $bestTransformationSet$ **do**
  $w_{(i,after)} \leftarrow w_{(i,before)} \times -1$;
  $T_{after} \leftarrow T_{before} - w_{(i,before)}$;
end

Algorithm 1: Threshold Value Optimization Strategy (TVOS).

B. Critical wire rewiring considering negative weights

Our work also considers the effects of negative weights in the critical wire rewiring. When negative weights are present, they alter the conditions under which wires are deemed critical. We establish the following criteria for identifying a critical wire in a TLG.

Theorem 1: Given a TLG represented by the weight-threshold vector $[w_1, w_2, \ldots, w_n; T]$, an input x_j with corresponding weight w_j is considered critical if either of the following conditions is met:

Condition 1: $w_j^+ > 0$, and the summation of all other positive weights, denoted as $\sum_{i=1, i\neq j}^{n} w_i^+$, is less than the threshold value T, i.e., $\sum_{i=1, i\neq j}^{n} w_i^+ < T$.

Condition 2: $w_j^- < 0$, and the summation of all positive weights and the negative weight w_j is less than the threshold value T, i.e., $(\sum_{i=1}^{n} w_i^+) + w_j^- < T$.

Let us explain the validity of Theorem 1 in the following paragraphs. Before explaining Theorem 1, we first define an input x_j satisfying either Condition 1 or Condition 2 of Theorem 1 as a *critical input*, and its corresponding weight w_j as a *critical weight*. Condition 1 implies that even with

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

all other positive weights are activated and all other negative weights are deactivated, the threshold value T cannot be reached without the contribution of the positive weight w_j^+. Therefore, to prevent the TLG from always outputting 0, x_j must be 1. Condition 2 implies that even with all positive inputs activated and all other negative weights are deactivated, the presence of negative weight w_j^- prevents the threshold value T from being reached. Therefore, to prevent the TLG from always outputting 0, x_j must be 0. In this work, it is crucial that the *critical input* must remain active in Condition 1 where its contribution is positive, while being deactivated in Condition 2 where its contribution is negative.

After identifying the critical wire, we limit the TLG propagation condition by adding an AND gate, as mentioned in Section II-E. However, for Condition 2, an adjustment is necessary. We need to set the *critical input* x_j to 0; therefore, a NOT gate is placed on x_j before it is connected to the AND gate.

To remove the impact of critical wire rewiring on TLG, we discuss these two conditions separately. In Condition 1, when the *critical input* x_j is set to 1, the TLG condition changes from $x_1w_1 + ... + x_jw_j^+ + ... + x_nw_n \geq T$ to $x_1w_1 + ... + 1 \bullet w_j^+ + ... + x_nw_n \geq T$. Therefore, when rewiring the critical wire, we need to adjust the threshold value to $T - w_j^+$ to maintain the original TLG. In Condition 2, when x_j is set to 0, the TLG condition changes from $x_1w_1 + ... + x_jw_j^- + ... + x_nw_n \geq T$ to $x_1w_1 + ... + 0 + ... + x_nw_n \geq T$. Thus, when rewiring the critical wire, the value of T does not need to be changed.

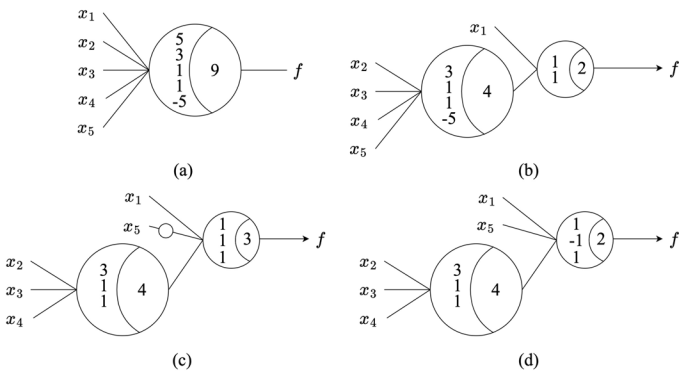

Figure 6: (a) A TLG. (b) A TLN after applying the critical wire rewiring Condition 1. (c) A TLN after applying the critical wire rewiring Condition 2. (d) A TLN after applying the transformation.

Figure 6 illustrates the process of conducting consecutive critical wire rewirings in a TLG. First in Figure 6(a), x_1 is identified as a critical wire for Condition 1 because $w_1 = 5 > 0$ and $3 + 1 + 1 < 9$. Therefore, we rewire x_1 by adding an AND gate and modify the threshold value from 9 to 4 $(= 9 - 5)$ as shown in Figure 6(b). Then, x_5 is identified as a critical wire for Condition 2 because $w_5 = -5 < 0$ and $3 + 1 + 1 - 5 < 4$. Therefore, we rewire x_5 by adding a NOT gate, then connecting to the AND gate as shown in Figure 6(c). The threshold value does not need to be changed in this condition. Applying the transformation on the NOT gate can further reduced the area cost as shown in Figure 6(d). Using the area cost formula $(\sum_{i=1}^{n} |w_i| + |T|)$, we can observe that in this example, the area cost is reduced from the original value of 24 to the new value of 14.

C. 2-*TLG substitution*

By employing the 2-TLG substitution method, 2-TLG can be treated as 1-TLG, allowing us to leverage techniques from Sections III-A and III-B. Algorithm 2 details the weight substitution process, which induces 1^{st}-order input x_k associated with w_k.

Input: 2-TLG $[w_{(1,2^{nd})}, ..., w_{(n,2^{nd})}, w_{((1,2),2^{nd})}, ...w_{((n-1,n),2^{nd})}; T_{2^{nd}}]$
Output: 1-TLG $[w_{(1,1^{st})}, ..., w_{(n,1^{st})}, ..., w_{(k,1^{st})}; T_{1^{st}}]$
for $i = 1$ *to* n **do**
  $w_{(i,1^{st})} \leftarrow w_{(i,2^{nd})}$;
end
$k \leftarrow n + 1$;
for $i = 1$ *to* $n - 1$ **do**
  for $j = i + 1$ *to* n **do**
    if $w_{((i,j),2^{nd})} \neq 0$ **then**
      $w_{(k,1^{st})} \leftarrow w_{((i,j),2^{nd})}$;
      $k \leftarrow k + 1$;
    end
  end
end
$T_{1^{st}} \leftarrow T_{2^{nd}}$

Algorithm 2: 2-TLG substitution.

Consider a 4-input 2-TLG represented as $[1, 1, 1, 1, -1, 0, 0, 0, 0, 0; 3]$, as shown in Figure 1(b). In accordance with Algorithm 2, the initial four weights $[1, 1, 1, 1]$ are transformed to construct a new 1-TLG. Subsequently, for the remaining $\binom{4}{2}$ 2-weights, the algorithm substitutes the non-zero weights with the newly allocated weights. In this example, only $w_{(1,2)}$ is a non-zero 2-weight. Therefore, the final 1-TLG can be represented as a 5-input TLG, denoted as $[1, 1, 1, 1, -1; 3]$.

D. *Proposed area-oriented TLS*

Figure 7 illustrates the flowchart of the proposed area-oriented TLS. Given a TLN to be optimized, it starts by selecting a target TLG. Then it conducts cost minimization, focusing on reducing the area cost of the TLG by minimizing the magnitude of threshold value. Next, during the critical wire rewiring stage, it continuously assesses whether there are available critical wires that can further reduce the area cost for a given TLG. Once all the TLGs have been selected as the target TLGs, the algorithm returns a synthesized area-oriented TLN.

Although negative weights have better advantages in RTDs, there are other technologies (Charge-Timed Threshold Logic and Self-Timed Threshold Logic) that also calculate area cost using the same model. The proposed area-oriented TLS is effective in these technologies as well.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

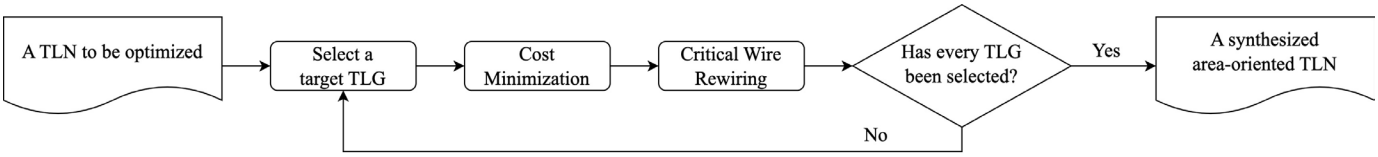

Figure 7: The flowchart of the proposed area-oriented threshold logic network synthesis algorithm.

IV Experimental Results

To evaluate the effectiveness of the proposed area-oriented TLS, we conducted a series of experiments and compared the results with those of previous works [13, 17, 25]. The proposed area-oriented TLS was implemented in C++ and the experiments were performed on a 2.9 GHz Linux platform (CentOS 7.9). The benchmarks used in the experiments are IWLS 2005 [8]. The input circuits to our area-oriented TLS are the results synthesized by previous works [13, 17, 25]. We obtained the executable files provided by the previous works and generated these TLNs as inputs in our environment. The input circuits have two versions: 6-input TLNs and 15-input TLNs. Both of them were optimized using two different strategies: Summation-oriented (minimizing the summation of $\sum |w_i| + |T|$ in all TLGs in the TLN) and Gate-oriented (minimizing the total gate count in the TLN) by previous works [13, 17, 25]. In all the experiments, we calculated the area cost using EQ(4).

Tables I, II, and III present detailed comparisons of our area-oriented TLS results with those of previous works [17], [13], and [25], respectively. Each table presents the area cost reduction achieved by applying our TLS. These tables share a consistent structure, with the columns defined as follows: Column 1 lists the benchmarks. Columns 2-7 and 8-13 present results for 6-input (TLNK6) and 15-input (TLNK15) TLNs, respectively. Columns 2 and 8 show area cost from previous work using the Summation-oriented strategy. Columns 3 and 9 present the area cost after applying our area-oriented TLS to these TLNs. Columns 4 and 10 show the corresponding area cost reduction. Similarly, Columns 5 and 11 show area cost from the previous work using the Gate-oriented strategy; Columns 6 and 12 present the area cost after applying our TLS; and Columns 7 and 13 show the area cost reduction. The final row of Columns $4, 7, 10$, and 13 displays the average reduction ratio considering all the benchmarks.

According to Tables I, II, and III, our approach consistently reduces area cost as compared to prior works across all TLNs. Notably, even in the worst cases (e.g., *pci_conf_cyc* in Table I), our approach does not increase area cost.

Table IV summarizes the reduction ratios across all TLNs using our approach as compared to the prior works [17][13][25]. Column 2 shows the TLN versions (6-input or 15-input TLNs). Column 3 shows the optimization strategy (Summation-oriented or Gate-oriented) used in the previous works. The last column shows the reduction ratios, extracted from the final rows of Tables I, II, and III. According to Table IV, it is evident that our approach demonstrates large reductions when applied to 15-input TLNs compared to 6-input TLNs. This is because TLGs with more inputs tend to have larger weights and threshold values, and our approach offers greater benefits for circuits with these characteristics. Similarly, our approach achieves a greater reduction ratio in Gate-oriented TLNs. This is because TLNs synthesized with the goal of minimizing gate counts tend to have more complex TLGs, which in turn leads to increased weights and threshold values. In summary, our approach reduces area cost for all the TLN versions and significantly lowers the area cost of TLNs consisting of TLGs with larger inputs.

V Conclusion

This paper introduces the first approach to effectively utilize negative weights within TLGs. Cost minimization using P-N weights transformation does not impair the expressiveness of a TLG, preserving the advantages of TLN's expressive power. Furthermore, our approach minimizes area costs across all TLN versions, demonstrating remarkable effectiveness, particularly for TLNs consisting of TLGs with large inputs. Our approach is applicable to both 1-TLG and 2-TLG structures.

References

[1] T. Akeyoshi, K. Maezawa, and T. Mizutani, "Weighted sum threshold logic operation of mobile (monostable-bistable transition logic element) using resonant-tunneling transistors," *IEEE Electron Device Lett.*, vol. 14, no. 10, pp. 475–477, 1993.

[2] M. J. Avedillo, J. M. Quintana, H. Pettenghi, P. Kelly, and C. Thompson, "Multi-threshold threshold logic circuit design using resonant tunnelling devices," *Electron. Lett.*, vol. 39, no. 21, pp. 1502–1504, 2003.

[3] V. Beiu, J. M. Quintana, and M. J. Avedillo, "Vlsi implementations of threshold logic-a comprehensive survey," *IEEE Trans. on Neural Netw.*, vol. 14, no. 5, pp. 1217–1243, 2003.

[4] Y.-C. Chen, R. Wang, and Y.-P. Chang, "Fast synthesis of threshold logic networks with optimization," in *Proc. of ASP-DAC*, 2016, pp. 486–491.

[5] L. Gao, F. Alibart, and D. B. Strukov, "Programmable cmos/memristor threshold logic," *IEEE Trans. on Nanotechnol.*, vol. 12, no. 2, pp. 115–119, 2013.

[6] A. Ghosh, A. Jain, and S. K. Sarkar, "Design and simulation of single electron threshold logic gate based programmable logic array," *Procedia Technol.*, vol. 10, pp. 866–874, 2013.

[7] T. Gowda, S. Vrudhula, and G. Konjevod, "A non-ILP based threshold logic synthesis methodology," in *Proc. of IWLS*, 2007, pp. 222–229.

[8] International Workshop on Logic Synthesis, "Iwls 2005 benchmarks," 2005. [Online]. Available: https://iwls.org/iwls2005/benchmarks.html

[9] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold logic circuits," in *Proc. of ICCAD*, 2011, pp. 396–403.

[10] C.-T. Lee, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Approximate

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

Table I: The results comparison between [17] and ours.

	TLNK6						TLNK15					
	Summation			Gates			Summation			Gates		
Circuits	[17]	Ours	R	[17]	Ours	R	[17]	Ours	R	[17]	Ours	R
ac97_ctrl	42791	39978	2813	45746	41341	4405	42631	39753	2878	46809	42678	4131
aes_core	78747	70656	8091	83144	71741	11403	80251	72234	8017	84466	75138	9328
alu4	2769	2421	348	2734	2316	418	2730	2369	361	2825	2380	445
apex6	2788	2563	225	2649	2306	343	2816	2578	238	2694	2310	384
C1355	1648	1617	31	1670	1638	32	1648	1617	31	1705	1654	51
C1908	1392	1304	88	1556	1426	130	1435	1350	85	1458	1357	101
C5315	4939	4488	451	4970	4348	622	4928	4494	434	5137	4447	690
C6288	9166	8314	852	10618	9196	1422	9851	8908	943	10456	8919	1537
C7552	5006	4710	296	5213	4766	447	4865	4582	283	5568	5032	536
dalu	3520	3129	391	3911	3314	597	3611	3188	423	3644	3116	528
des_area	16558	15558	1000	17058	15872	1186	11563	10672	891	16026	14749	1277
des	12006	11004	1002	12399	10881	1518	17740	16570	1170	11723	10372	1351
frg2	2354	2068	286	2426	2035	391	2354	2068	286	2496	2124	372
i10	6186	5647	539	6374	5625	749	6202	5661	541	6785	5981	804
i2c	3164	2944	220	3175	2801	374	3274	3017	257	3296	2888	408
k2	3458	3115	343	3568	3077	491	3377	3085	292	3957	3429	528
mem_ctrl	23891	22540	1351	24275	22225	2050	24108	22672	1436	24307	22305	2002
pair	5400	4869	531	5442	4797	645	5614	5032	582	5380	4696	684
pci_bridge32	54410	52544	1866	59577	55930	3647	55912	53927	1985	63799	59495	4304
pci_conf_cyc	316	316	0	316	316	0	316	316	0	316	316	0
pci_spoci_ctrl	2259	2100	159	2519	2197	322	2270	2119	151	2277	2087	190
rot	1842	1700	142	2030	1808	222	1815	1680	135	1951	1734	217
s13207	8790	8349	441	8831	8093	738	8530	8134	396	9303	8582	721
s9234	5088	4731	357	5200	4628	572	5187	4759	428	5845	5153	692
sasc	2140	1975	165	2573	2272	301	2140	1975	165	2431	2122	309
simple_spi	2892	2586	306	2976	2615	361	2895	2586	309	3243	2862	381
spi	10374	9574	800	10260	9257	1003	10396	9597	799	10620	9560	1060
stepperm	550	512	38	534	509	25	550	512	38	628	591	37
systemcaes	28275	26446	1829	29616	26447	3169	28205	26102	2103	28914	26160	2754
systemcdes	11343	10097	1246	11207	9780	1427	11011	9794	1217	10597	9287	1310
t481	596	527	69	601	501	100	575	507	68	578	490	88
tv80	23965	21540	2425	24929	22184	2745	23893	21399	2494	26127	22711	3416
usb_funct	47512	41946	5566	47677	40760	6917	47761	41938	5823	48922	41927	6995
usb_phy	1501	1330	171	1656	1378	278	1492	1320	172	1716	1411	305
wb_conmax	115814	111141	4673	115929	108709	7220	114500	110098	4402	136072	121628	14444
x3	2487	2300	187	2444	2144	300	2442	2292	150	2665	2346	319
Summation	509981	475961	34020	520057	467892	52165	469031	436943	32088	510270	456899	53371
Ratio			**7.20%**			**10.03%**			**7.33%**			**11.97%**

Table II: The results comparison between [13] and ours.

	TLNK6						TLNK15					
	Summation			Gates			Summation			Gates		
Circuits	[13]	Ours	R	[13]	Ours	R	[13]	Ours	R	[13]	Ours	R
ac97_ctrl	27651	26505	1146	43564	39246	4318	27620	26523	1097	88500	74804	13696
aes_core	49497	47781	1716	83477	74560	8917	49587	47931	1656	117160	102385	14775
alu4	1623	1553	70	2818	2472	346	1635	1565	70	5340	4468	872
apex6	1571	1507	64	2474	2241	233	1616	1531	85	3651	3354	297
C1355	1024	1024	0	1606	1566	40	1024	1024	0	2155	2049	106
C1908	946	936	10	1461	1376	85	925	917	8	2341	2013	328
C5315	3441	3241	200	4964	4503	461	3433	3235	198	6240	5578	662
C6288	5133	5121	12	10992	9876	1116	5058	5047	11	14864	12774	2090
C7552	3623	3577	46	5270	4931	339	3652	3595	57	7247	6686	561
dalu	2036	1969	67	3817	3334	483	2039	1981	58	5712	4881	831
des_area	11329	10049	1280	16732	15902	830	8107	7874	233	15470	13789	1681
des	7962	7759	203	12380	11014	1366	11520	10225	1295	31867	27461	4406
frg2	1822	1576	246	2244	1892	352	1822	1576	246	6069	5030	1039
i10	4163	4066	97	6030	5459	571	4184	4077	107	11686	9895	1791
i2c	2066	2066	0	2941	2709	232	2065	2065	0	4128	3920	208
k2	2252	2215	37	3433	3087	346	2200	2162	38	8290	7071	1219
mem_ctrl	18116	17687	429	23438	21645	1793	18037	17615	422	34791	31266	3525
pair	3290	3138	152	5110	4624	486	3269	3119	150	8093	6894	1199
pci_bridge32	41692	40659	1033	59635	56357	3278	41803	40799	1004	81494	75498	5996
pci_conf_cyc	283	262	21	316	295	21	283	262	21	316	295	21
pci_spoci_ctrl	1528	1528	0	2389	2159	230	1564	1564	0	4260	3649	611
rot	1232	1193	39	1933	1747	186	1236	1195	41	3048	2652	396
s13207	6546	6492	54	8319	7708	611	6446	6400	46	13554	12810	744
s9234	3762	3645	117	5128	4645	483	3802	3666	136	10040	8628	1412
sasc	1606	1526	80	2466	2183	283	1606	1526	80	3906	3394	512
simple_spi	2039	2039	0	2876	2531	345	2039	2039	0	6086	5272	814
spi	6778	6778	0	9685	8957	728	6767	6767	0	16970	14870	2100
stepperm	271	270	1	495	470	25	276	275	1	854	685	169
systemcaes	20987	20475	512	29870	27185	2685	21170	20603	567	33347	30363	2984
systemcdes	6451	6197	254	10657	9625	1032	6216	5980	236	16949	15145	1804
t481	355	343	12	618	539	79	356	338	18	1025	911	114
tv80	16336	15774	562	24118	21453	2665	16317	15741	576	42758	37306	5452
usb_funct	30105	29635	470	44743	39029	5714	30196	29681	515	63850	55025	8825
usb_phy	978	944	34	1412	1214	198	976	945	31	2030	1623	407
wb_conmax	100848	99004	1844	113493	108524	4969	97957	96089	1868	172762	157244	15518
x3	1379	1364	15	2242	2013	229	1379	1364	15	3219	2932	287
Summation	363070	353393	9677	553146	507071	41757	360562	350773	9789	761572	677816	83756
Ratio			**2.67%**			**7.55%**			**2.71%**			**11.00%**

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.

Table III: The results comparison between [25] and ours.

	TLNK6						TLNK15					
	Summation			Gates			Summation			Gates		
Circuits	[25]	Ours	R	[25]	Ours	R	[25]	Ours	R	[25]	Ours	R
ac97_ctrl	27651	26505	1146	41027	37371	3656	27620	26523	1097	58720	52132	6588
aes_core	47823	46411	1412	64974	60553	4421	48378	46946	1432	70749	65450	5299
alu4	1619	1549	70	2135	1950	185	1635	1565	70	2768	2500	268
apex6	1571	1507	64	2110	1977	133	1616	1531	85	2424	2290	134
C1355	1024	1024	0	1466	1458	8	1024	1024	0	2015	1936	79
C1908	942	932	10	1238	1201	37	925	917	8	1542	1441	101
C5315	3441	3241	200	4125	3859	266	3433	3235	198	4559	4216	343
C6288	5126	5115	11	8911	8350	561	5058	5047	11	9759	8915	844
C7552	3603	3557	46	4448	4262	186	3620	3563	57	4965	4761	204
dalu	2036	1969	67	2760	2522	238	2039	1981	58	3208	2975	233
des_area	11325	10046	1279	13072	12771	301	8084	7858	226	10020	9435	585
des	7925	7735	190	9301	8707	594	11516	10221	1295	17869	16839	1030
frg2	1822	1576	246	2081	1780	301	1822	1576	246	3314	2793	521
i10	4151	4055	96	5010	4714	296	4176	4070	106	7226	6601	625
i2c	2066	2066	0	2493	2345	148	2065	2065	0	3275	3166	109
k2	2252	2215	37	2789	2625	164	2200	2162	38	4330	4099	231
mem_ctrl	18116	17687	429	21043	19819	1224	18037	17615	422	23170	21728	1442
pair	3286	3134	152	4096	3839	257	3265	3115	150	4781	4364	417
pci_bridge32	41517	40571	946	55109	52829	2280	41620	40703	917	68415	65336	3079
pci_conf_cyc	283	262	21	316	295	21	283	262	21	316	295	21
pci_spoci_ctrl	1528	1528	0	2065	1921	144	1564	1564	0	2905	2656	249
rot	1232	1193	39	1650	1540	110	1236	1195	41	1883	1735	148
s13207	6490	6439	51	7514	7116	398	6386	6343	43	11486	11027	459
s9234	3762	3645	117	4307	4043	264	3802	3666	136	6101	5756	345
sasc	1606	1526	80	2225	1986	239	1606	1526	80	2677	2455	222
simple_spi	2039	2039	0	2551	2301	250	2039	2039	0	4483	4033	450
spi	6778	6778	0	7732	7373	359	6767	6767	0	11179	10450	729
stepperm	271	270	1	445	434	11	276	275	1	522	478	44
systemcaes	20499	19995	504	24665	23077	1588	20290	19789	501	26869	25194	1675
systemcdes	6237	6026	211	8765	8107	658	6053	5847	206	10302	9625	677
t481	351	339	12	461	423	38	352	334	18	530	494	36
tv80	16331	15770	561	19300	17782	1518	16312	15737	575	25552	23557	1995
usb_funct	30105	29635	470	36798	33096	3702	30196	29681	515	48354	42805	5549
usb_phy	978	944	34	1143	1034	109	976	945	31	1341	1166	175
wb_conmax	97624	96320	1304	95753	93823	1930	94934	93674	1260	118383	112424	5959
x3	1379	1364	15	1916	1768	148	2105	2004	101	2312	2179	133
Summation	384789	374968	9821	424767	401680	23087	383310	373365	9945	519584	485174	34410
Ratio			**2.55%**			**5.44%**			**2.59%**			**6.62%**

Table IV: Summary of an experimental results on reduction ratio of area cost.

Methods	Version	Opt. Strategy	Red. Ratio
[17]	TLNK6	Summation	7.20%
		Gate	10.03%
	TLNK15	Summation	7.33%
		Gate	**11.97%**
[13]	TLNK6	Summation	2.67%
		Gate	7.55%
	TLNK15	Summation	2.71%
		Gate	11.00%
[25]	TLNK6	Summation	2.55%
		Gate	5.44%
	TLNK15	Summation	2.59%
		Gate	6.62%

logic synthesis by genetic algorithm with an error rate guarantee," in *Proc. of ASP-DAC*, 2023, pp. 146–151.

[11] M.-J. Li, Y.-C. Yen, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "A constructive approach for threshold function identification," *ACM TODAES*, vol. 28, no. 5, pp. 1–19, 2023.

[12] C.-C. Lin, C.-W. Huang, C.-Y. Wang, and Y.-C. Chen, "In&out: Restructuring for threshold logic network optimization," in *Proc. of ISQED*, 2017, pp. 413–418.

[13] C.-C. Lin, C.-S. Lin, Y.-H. Tsai, Y.-C. Chen, and C.-Y. Wang, "Don't care computation and de morgan transformation for threshold logic network optimization," *IEEE TCAD*, vol. 41, no. 5, pp. 1412–1422, 2022.

[14] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for threshold logic circuit minimization," in *Proc. of DATE*. IEEE, 2014, pp. 1–6.

[15] C.-H. Liu, C.-C. Lin, Y.-C. Chen, C.-C. Wu, C.-Y. Wang, and S. Yamashita, "Threshold function identification by redundancy removal and comprehensive weight assignments," *IEEE TCAD*, vol. 38, no. 12, pp. 2284–2297, 2018.

[16] S. Muroga, *Threshold logic and its applications*, 1971.

[17] A. Neutzling, J. M. Matos, A. Mishchenko, A. Reis, and R. P. Ribas, "Effective logic synthesis for threshold logic circuit design," *IEEE TCAD*, vol. 38, no. 5, pp. 926–937, 2019.

[18] G. Papandroulidakis, A. Serb, A. Khiat, G. V. Merrett, and T. Prodromakis, "Practical implementation of memristor-based threshold logic gates," *IEEE TCAS-I*, vol. 66, no. 8, pp. 3041–3051, 2019.

[19] H. Pettenghi and J. Q. MJ Avedillo, "Improved nanopipelined rtd adder using generalized threshold gates," *IEEE Trans. on Nanotechnol.*, vol. 10, no. 1, pp. 155–162, 2009.

[20] H. Pettenghi, M. J. Avedillo, and J. M. Quintana, "A novel contribution to the rtd-based threshold logic family," in *Proc. of ISCAS*, 2008, pp. 2350–2353.

[21] C.-K. Tsai, C.-Y. Wang, C.-Y. Huang, and Y.-C. Chen, "Sensitization criterion for threshold logic circuits and its application," in *Proc. of ICCAD*, 2013, pp. 226–233.

[22] S. Vrudhula, N. Kulkami, and J. Yang, "Design of threshold logic gates using emerging devices," in *Proc. of ISCAS*, 2015, pp. 373–376.

[23] Y.-C. Yen, M.-J. Li, Y.-T. Li, Y.-C. Chen, I. Chen, and C.-Y. Wang, "9-input threshold function identification using a new necessary condition of threshold function," *IEEE TCAD*, vol. 43, no. 12, pp. 4676–4686, 2024.

[24] R. Zhang, P. Gupta, L. Zhong, and N. Jha, "Synthesis and optimization of threshold logic networks with application to nanotechnologies," in *Proc. of DATE*, vol. 2, 2004, pp. 904–909.

[25] L.-C. Zheng, H.-J. Chang, Y.-C. Chen, and J.-Y. Jou, "1st-order to 2nd-order threshold logic gate transformation with an enhanced ilp-based identification method," in *Proc. of ASP-DAC*, 2021, pp. 469–474.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on June 01,2026 at 02:39:43 UTC from IEEE Xplore. Restrictions apply.