2026 Design, Automation & Test in Europe Conference (DATE 2026)

Advancing LUT-based Threshold Logic Synthesis with Enhanced Area Estimation

Yu-Shan Lin[1] and Yung-Chih Chen[1,2]
[1]*Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan*
[2]*Arculus System Co., Ltd., Hsinchu, Taiwan*
{m11107431, ycchen.ee}@mail.ntust.edu.tw

***Abstract*—Threshold logic has regained research interest, leading to the development of design automation techniques. A recent lookup table (LUT)-based threshold logic synthesis method has shown promising results by leveraging the strengths of LUT mapping. However, its reliance on the disjoint support decomposition (DSD) manager for caching NPN-equivalent functions can lead to inaccurate area estimation, as it overlooks key properties of threshold functions. This misestimation degrades overall synthesis quality. To address this, we propose improvements that enhance area estimation by extending the DSD manager and correcting function complementation in the covering process. These enhancements allow the mapper to select lower-cost coverings more effectively. Experimental results show an average area reduction of 6.12% with cut size 6 and 6.54% with cut size 15, compared to the original LUT-based method.**

I. Introduction

Threshold logic [1] was introduced early on and has regained considerable attention in recent years. As a Boolean logic representation, a threshold logic gate (TLG) can implement more complex functions than conventional primitive gates. Moreover, its neuron-like behavior, which determines its output based on the weighted sum of its inputs, makes it well-suited for realization with emerging technologies [2], [3] and finds applications in machine learning [4].

Like a neuron, a TLG takes several input signals, applies specific weights to these inputs, and sums them. If the weighted sum equals or exceeds a predefined threshold, the gate outputs a signal, typically a binary 1. If the threshold is not met, the output is 0.

The function of a TLG with n inputs is defined as follows:

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

where $w_1 \sim w_n$ denote the weights and T denotes the threshold, which are typically integers. A Boolean function that can be realized using a single TLG is also called a threshold function (TF). A TF $f(x_1, x_2, \ldots, x_n)$ also can be represented with a weight-threshold vector $[w_1, w_2, \ldots, w_n; T]$.

This work was supported by the National Science and Technology Council, Taiwan, under grants NSTC 111-2221-E-011-137-MY3 and 113-2640-E-011-003.

The area cost of a TLG is typically estimated by the sum of the absolute values of its weights and threshold, as follows:

$$TLG\ cost = \sum_{i=1}^{n}(|w_i|) + |T| \quad (2)$$

A threshold logic network (TLN) is a logic network composed of TLGs, which can be used to implement Boolean functions. A conventional Boolean logic network can be transformed into a TLN through suitable synthesis methods [2], [5], [6]. The state-of-the-art (SOTA) synthesis method [6] leverages the lookup table (LUT)-based technology mapper [7] to achieve TLG-oriented technology mapping. The core methodology involves exploiting a TF identification heuristic [8] and preserving only cut functions that are TFs during mapping. This LUT-based method quickly surpassed prior synthesis methods [2], [5], [9], [10] due to its superior quality and performance. However, subsequent optimization approaches [11]–[13] significantly enhanced the TLNs generated by the LUT-based method, leading us to question whether the method's quality still has room for improvement.

Through in-depth analysis of the LUT-based method, we identified three key limitations. First, the method overlooks the influence of the complemented inputs on the area cost of a TF function. Second, the method exploits the disjoint-support decomposition (DSD) manager [14], [15] to cache NPN-equivalent cut functions for saving TF identifications. However, this may lead to incorrect area cost estimation when two NPN-equivalent TFs have different area costs. Third, the strategy for determining whether a cut function is complemented during covering is insufficiently effective, leading to inaccurate area cost estimation.

Thus, in this paper, we propose enhancements to the LUT-based method to address the issues. The first limitation can be easily tackled by applying a weight transformation before computing the TLG cost. For the second limitation, we extend the DSD manager to mitigate the aliasing problem. For the last limitation, we propose a new strategy based on the DSD representation of the cut function.

We conducted experiments on a set of benchmark circuits from the IWLS 2005 benchmark suite [16]. The experimental results show that for TLN synthesis with a cut size of $k = 6$, our method achieves an average area reduction of 6.12% compared to the original LUT-based method. When the cut size is set to 15, the achieved average area reduction is 6.54%.

978-3-9826741-1-7/DATE26/© 2026 EDAA

Furthermore, a recent study [17] proposed an integer linear programming (ILP)-based approach for identifying 2nd-order TFs (2-TFs) and showed that a 2nd-order TLG (2-TLG) can realize certain functions with lower area cost than a conventional TLG. Building on this, [18] presented a method to transform a conventional TLN into a lower-cost 2nd-order (2-TLN) by primarily converting the TLGs individually. This approach was later improved in [19]. However, a direct synthesis method from Boolean logic to a 2-TLN remains unavailable. To address this, we adapt the LUT-based TLN synthesis framework [6] for 2-TLN synthesis by replacing its internal TF identification heuristic with the ILP-based 2-TF identification from [17]. Nonetheless, we observe that this approach does not consistently yield lower-cost results, as NPN-equivalent functions exhibit greater area variation in 2-TLG implementations. To overcome this limitation, we apply the proposed enhancements. Experimental results show that our method achieves an average area reduction of 8.2% compared to the baseline LUT-based 2-TLN synthesis.

The contributions of this work are summarized as follows:

- We propose two enhancements for more accurate area estimation in TLN synthesis. The first extends the DSD manager to handle TLGs with negative weights, improving estimation accuracy. The second selectively complements cut functions before area evaluation to avoid overestimation.
- We adapt the LUT-based TLN synthesis method for 2-TLN synthesis. To the best of our knowledge, this is the first work to directly synthesize a 2-TLN from a conventional Boolean logic network.

The rest of this paper is organized as follows: Section II reviews relevant background. Section III discusses the motivation behind this work. Section IV introduces the proposed enhancements. Section V shows the experimental results. Section VI concludes this work.

II. Preliminaries

A. Threshold Function Identification

The task of TF identification is to check whether a given Boolean function can be realized with a single TLG. It is essentially a process of finding the weights and the threshold of a TLG for the Boolean function. Thus, if the Boolean function is identified as a TF, the corresponding weights and threshold are obtained.

The most widely used method for identifying TFs is the effective heuristic proposed in [8]. This method first converts inputs with negative polarities to positive, if necessary, and then checks whether the given function is positive unate, as only unate functions can qualify as TFs. The next step involves constructing a system of inequalities, where the variables represent weights, based on the function's on-set and off-set. The heuristic then incrementally adjusts the variable values to solve the inequalities. If a solution is found, the function is classified as a TF, after which the weight values are obtained, and the threshold value is calculated. The work [20] further improves the heuristic by eliminating redundant inequalities and proposing a more effective value adjustment strategy.

In addition to heuristic methods, early research [2] introduced an exact approach that formulates TF identification as an ILP problem, enabling the computation of the minimum weights and threshold.

TF identification is central to TLN synthesis and is performed repeatedly throughout the process. Consequently, it directly impacts both the quality and performance of the TLN synthesis method.

B. LUT-based TLN Synthesis

FPGA technology mapping aims to minimize the number of LUTs while achieving near-optimal delay, improving area efficiency, and selecting an effective final cover [7], [21]. The first TLN synthesis method based on this flow was proposed in [22]. It uses dynamic programming to enumerate all possible cuts from primary inputs (PIs) to primary outputs (POs) [21], [23], and identifies TF cuts using a heuristic from [8]. Only TF cuts are retained and the best ones are selected to construct the final TLN. To reduce redundant computation, the method leverages a DSD manager [14] that stores representatives of each NPN class in a shared tree and reuses previously computed TF cut results. However, the overall flow requires two passes of cut enumeration.

The up-to-date synthesis method [6] enhances this flow by performing TF identification during the cut enumeration and completing the covering task in the same stage. In this work, we build upon this method, focusing on improving the area estimation of a TF cut.

In addition to synthesis methods, some prior works proposed optimization methods, including rewiring [9], [12], node collapsing [13], [24], and interconnect reduction [11] to simplify TLNs.

C. Disjoint Support Decomposition Manager

In [14], the authors introduce a data structure and computation engine based on DSD [25], which is a special case of Boolean decomposition. The DSD structure (if exists) of a function is a tree of nodes with non-overlapping supports and optional complemented attributes on the edges. A DSD is said to be *full* if all logic nodes are elementary gates (AND, XOR, MUX). If a full DSD does not exist, the function can be represented by a decomposition tree consisting of some elementary gates, as well as one or more nodes that cannot be further decomposed by DSD, referred to as prime nodes [25], or non-DSD nodes.

The DSD manager [14] provides an efficient method for handling DSD structures, supporting Boolean operations, function checking, and caching intermediate computations. DSD structures stored in the DSD manager include essential information such as function IDs representing the canonical forms of the DSD structures, optional complemented output attributes, and PN-configurations for permuting or complementing inputs of the canonical forms to recover the functions. For instance, the function, $f = ca!b(d + eh)$, is represented as the DSD

structure AND(n, n, n, !AND(n, !AND(n, n))) with the PN-configuration $(c, a, !b, !d, e, h)$.

D. 2nd-Order Threshold Function

The function of an n-input 2-TLG is defined as follows:

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & if \sum_{i=1}^{n} w_i \cdot x_i + \\ & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j} \cdot x_i \cdot x_j \geq T \\ 0, & otherwise \end{cases} \quad (3)$$

In addition to the 1st-order weight w_i with respect to each input variable x_i, each pair of input variables (x_i, x_j) has a 2nd-order weight $w_{i,j}$, which is activated when both x_i and x_j equal 1. Similarly, if the weighted sum is not less than the threshold T, the function outputs 1; otherwise, 0. The area cost of a 2-TLG is defined as follows [17]:

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

where the cost of a 2nd-order weight is 4 times that of a 1st-order weight.

The work [17] proposes an exact ILP-based approach for 2-TF identification, i.e., determining whether a given Boolean function can be implemented with a single 2-TLG. The method formulates the identification as an ILP problem. It first enumerates the truth table of f and, for each input pattern, generates a constraint (inequality) according to Eq. (3). Since weights and the threshold can be negative, each is represented using two non-negative variables, w_x^+ and w_x^- (or T^+ and T^-), with the actual value given by $w_x^+ - w_x^-$ (or $T^+ - T^-$). Furthermore, one of the paired variables is forced to zero. The objective function is to minimize the area cost defined in Eq. (4). If the ILP solver finds a feasible solution, it also returns the weights and threshold; otherwise, f cannot be realized by a 2-TLG.

In this work, we adapt the LUT-based synthesis method [6] for 2-TLN synthesis by replacing the internal TF identification heuristic with this ILP-based 2-TF identification. Our intention is to show the importance of the proposed enhancements for 2-TLN synthesis.

III. Motivation

In the SOTA TLN synthesis method [6], the DSD manager is used to store functions along with their DSD structures, perform Boolean operations, and cache intermediate computations. However, a potential issue arises when estimating the area cost of a TLG with negative input variables. The TF identification heuristic transforms the cut function into an equivalent form with only positive input variables, disregarding the effect of negative weights associated with the original function. As a result, functions with the same DSD structure but differing in PN-configurations or complemented output attributes may be stored in the same entry in the DSD manager. This can lead to incorrect area cost estimation during synthesis.

For example, consider two functions: $F_{cut1} = !a!(!bc)$ and $F_{cut2} = !a!(bc)$. Both share the same DSD structure $F_{dsd} =$ AND(n, ! AND(n, n)), but differ in their PN-configurations: $\{!a, !b, c\}$ and $\{!a, b, c\}$, respectively. Using the heuristic, both are assigned the same area cost of 7 with the weight-threshold vector $[2, 1, 1; 3]$ and are stored in the same entry in the DSD manager. However, when negative weights are properly considered, their actual area costs are 4 (with $[-2, 1, -1; 0]$) and 5 (with $[-2, -1, -1; -1]$), respectively. Since the DSD manager caches only one entry per structure, storing one function may lead to incorrect area cost estimation for the other.

Additionally, during cut enumeration, the synthesis method determines whether a cut function should be complemented based on the complemented attributes of its fanins. However, we observe that these pre-determined functions may not match the actual functions selected during covering, leading to inaccurate area cost estimation.

To address the impact of negative weights, we apply the positive-negative weight transformation [26], converting positive weights on negative literals into equivalent negative weights during area computation. Furthermore, to mitigate aliasing issues in the DSD manager, we extend its functionality to distinguish functions with the same DSD structure but different configurations. Finally, we introduce a new strategy to decide whether a cut function should be complemented, using its DSD structure rather than relying solely on fanin attributes.

Since the weight transformation is not our contribution, we focus on the latter two enhancements in the following section.

IV. Proposed Enhancements

A. Enhancement 1: DSD Manager Extension

To mitigate aliasing issues in the DSD manager, we enhance it to accurately estimate the area costs of TF functions that share the same DSD structure but differ in the number of negative input variables or have different optional complemented output attributes.

In addition to the original information stored in the DSD manager, such as function IDs, DSD structures, output negations, PN-configurations, and area costs, we introduce two additional components to better distinguish functions with differing area costs. The first component records the counts of positive unate, negative unate, and binate variables (#PNB), based on the observation that functions with the same number of negative inputs tend to exhibit similar area costs. However, this alone is insufficient, as different negative weight values can lead to varying threshold values. Therefore, the second component, termed the negation distribution, captures the number of negative input variables occurring at different levels of the DSD structure, enabling finer-grained classification for accurate area cost estimation.

To compute the negation distribution, we classify cut functions into three categories: single-function, multi-sub-function, and PRIME-function. A cut function is considered a single-function if every internal node in its DSD structure has at most one child that is not an input variable. In contrast, it is classified as a multi-sub-function if at least one internal

node has two or more children that are themselves internal nodes. Any cut function with a valid DSD structure thus falls into either the single-function or multi-sub-function category. Finally, a function is categorized as a PRIME-function if its decomposition tree contains prime (i.e., non-decomposable) nodes.

Example 1: Fig. 1 illustrates the extended DSD manager handling four single-function class functions that share the same DSD structure but have different area costs and are initially absent from the DSD manager. Consider the first function, $F_1 = (a!(!d!(!c!be)))$, whose DSD structure is AND(n, !AND(n, !AND(n, n, n))). Since F_1 is not yet in the DSD manager, a new entry is created with an ID of 4 (as highlighted).

Next, we determine the #PNB (Positive, Negative, Binate) classification: F_1 has 3 positive unate, 2 negative unate, and 0 binate variables, leading to a #PNB entry of 3P2N0B, which is added and linked to the DSD structure entry.

Then, we compute the negation distribution. The DSD tree is 3 levels deep, and the two negative input variables $!c$ and $!b$ appear at the last level, resulting in a distribution of $\{0, 0, 2\}$. A corresponding entry with this distribution and an area cost of 15 is added to the negation distribution table and linked to the 3P2N0B entry.

Next, consider the second function $F_2 = (!a!(!e!(cb!d)))$. Since its DSD structure already exists in the DSD structure table (shared with F_1), no new DSD structure entry is created. F_2 also has 3 positive unate, 2 negative unate, and 0 binate input variables, matching F_1, and therefore shares the same 3P2N0B entry in the #PNB table.

However, its negation distribution differs. In F_2, the negated variables $!a$ and $!d$ appear at the first and the last levels of the DSD structure, respectively, resulting in a distribution of $\{1, 0, 1\}$. Thus, a new entry with this distribution and an area cost of 12 is added to the negation distribution table under the shared 3P2N0B entry.

Finally, consider the last two functions: $F_3 = (a!(!d!(!cbe)))$ and $F_4 = (a!(!e!(c!bd)))$. Both share the same DSD structure, #PNB classification (4P1N0B), and negation distribution $\{0, 0, 1\}$. Following the same procedure, we add an entry for 4P1N0B in the #PNB table and a corresponding negation distribution entry with an area cost of 16.

This example illustrates that, if the original DSD manager (without considering negative weights) were used, all four functions, despite their structural and semantic differences, would be assigned the same area cost of 17 (based on F_1). In contrast, the extended DSD manager correctly distinguishes between functions based on #PNB and negation distribution, enabling accurate area estimation and effective reuse of precomputed information when applicable, such as in the case of F_3 and F_4.

Example 2: Fig. 2 illustrates an example belonging to the multi-sub-functions class. To compute the #PNB, we apply the same method used for the single-function class. For computing the negation distribution, however, we adopt a hierarchical approach that recursively processes the structure from POs to PIs, determining the negation distribution of each sub-function.

Consider the function $F_5 = (!(!da)!(!b!c))$, which consists of two sub-functions $f_1 = !(!da)$ and $f_2 = !(!b!c)$. Since F_5

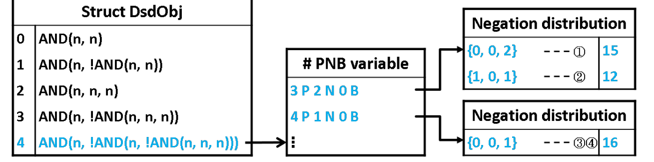

Fig. 1. Example 1 of extended DSD manager implementation.

Fig. 2. Example 2 of extended DSD manager implementation.

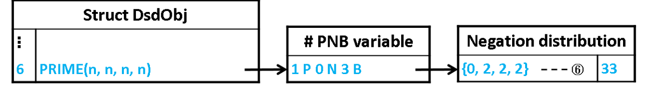

Fig. 3. Example 3 of extended DSD manager implementation.

is not yet present in the DSD manager, a new entry with ID 5 is created and highlighted. We then add an entry labeled 3P1N0B to the #PNB table, as F_5 contains 3 positive unate and 1 negative unate variables.

Next, the negation distribution is computed by analyzing the levels of the DSD structure. The DSD tree for F_5 has two levels, where each sub-function corresponds to a node. At the first level (the root), no negative input variables are present, resulting in a count of 0. At the second level, sub-function f_1 contains 1 negative input variable, while f_2 contains none. Therefore, the negation distribution for F_5 is $\{0, 1, 0\}$.

Finally, an entry for the negation distribution $\{0, 1, 0\}$ with an area cost of 12 is added to the negation distribution table and linked to the 3P1N0B entry in the #PNB table.

Example 3: Fig. 3 illustrates an example of the PRIME-function class. When a cut function contains prime nodes, it is rare for different functions to occupy the same storage space in the DSD manager, since such functions are not decomposable via DSD. In these cases, we directly compute the negation distribution based on the variable order.

Consider the function $F_6 = (d!a + c!b!a + !cb!a + !c!ba)$. Since F_6 is not found in the DSD manager, a new entry with ID 6 is created and highlighted. Given that it contains 1 positive unate, 0 negative unate, and 3 binate variables, an entry labeled 1P0N3B is added to the #PNB table and linked to the DSD structure entry.

To compute the negation distribution, we follow the variable order $(d \to b \to c \to a)$ and determine the unateness of each variable. Each variable is encoded as: 0 for positive unate, 1 for negative unate, and 2 for binate. Accordingly, the distribution $\{0, 2, 2, 2\}$ is obtained. A new entry with this distribution and an area cost of 33 is then added to the negation distribution table and linked to the 1P0N3B entry in the #PNB table.

By incorporating these two additional components, #PNB and negation distribution, the enhanced DSD manager provides more accurate area cost estimation and avoids redundant computations for structurally similar functions. For non-TF functions, we assign a large constant (e.g., 999999999) as the area cost to effectively exclude them from selection.

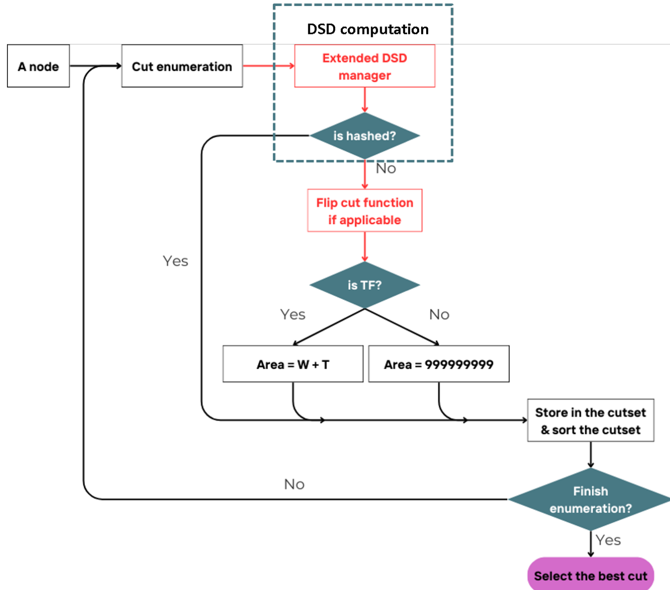

Fig. 4. Flow of finding the best cut for a given node.

B. *Enhancement 2: Cut Function Flipping Strategy*

As discussed earlier, the original method may heuristically complement a cut function during area cost estimation. However, the complemented version may not correspond to the function actually selected in the final cover, leading to inaccurate area cost estimation.

Through detailed analysis, we observe that for cut functions classified as single-function or multi-sub-functions, the version with an optional complemented output attribute set to 0 in the DSD manager is more likely to match the function ultimately selected. Based on this observation, we propose a refinement: when estimating the area cost, if a cut function belongs to one of these two classes and has its optional complemented output attribute set to 1, we flip (i.e., complement) the function before computing its area.

For example, consider the cut function $F_{dsd2} = !(!c!(ba))$, where the optional complemented output attribute is 1. We flip the function to obtain $F^{'}_{dsd2} = (!c!(ba))$. This transformation changes the area cost from 6 (with weight-threshold vector $[1, 1, 2; 2]$) to 5 (with vector $[-1, -1, -2; -1]$).

Importantly, this flipping is performed solely for more accurate area cost estimation and only affects the entries in the #PNB and negation distribution tables. The DSD structure and the core synthesis flow remain unchanged, preserving the correctness of the original method.

We evaluated this enhancement (Enhancement 2) on a set of benchmark circuits from the IWLS 2005 suite [16]. The evaluation shows a substantial reduction in mismatches between estimated and actually selected cut functions, with the average decreasing from 26.42% to just 0.02%. Due to space limitations, we omit the detailed results.

C. *Enhanced Flow for Best Cut Computation*

We integrate the proposed enhancements into the best cut computation flow described in [6], as illustrated in Fig. 4. A set of cuts is enumerated for the node under consideration. For each cut function, we first perform DSD computation. If the hash table lookup fails, the cut function is flipped when applicable. TF identification is then performed, and the area cost is computed. If the function is not a TF, a large constant is used as its area cost. Each cut function and its area cost are stored in the cut set and sorted. Finally, once cut enumeration is complete, the first entry in the sorted cut set is selected as the best cut.

TABLE I
EXPERIMENTAL RESULTS OF TLN SYNTHESIS WITH $k = 6$.

circuit	LUT-based method [6]				Ours				R_Cost (%)
	#TLG	D	Cost	Time (s)	#TLG	D	Cost	Time (s)	
pci_conf.	56	3	261	0.03	52	3	255	0.03	2.30
steppermoto.	74	15	260	0.12	75	17	256	0.12	1.54
usb_phy	221	7	903	0.21	210	7	856	0.20	5.20
sasc	347	9	1588	0.25	360	9	1465	0.29	7.75
pci_spoci_ctrl	414	14	1498	0.83	323	14	1355	0.74	9.55
simple_spi	605	17	2010	0.50	447	16	1662	0.55	17.31
i2c	611	19	2035	0.60	474	18	1936	0.72	4.86
systemcdes	1299	20	6220	3.39	1423	20	5438	5.58	12.57
spi	1950	23	6701	2.99	1505	22	6437	3.67	3.94
des_area	2353	23	11326	5.95	2410	23	10968	8.59	3.16
tv80	3864	37	16250	9.93	3822	36	15268	9.39	6.04
mem_ctrl	4483	26	17622	7.64	4107	26	17053	9.94	3.23
systemcaes	4805	29	20691	7.73	4846	31	19272	14.20	6.86
ac97_ctrl	6859	8	27425	4.87	6392	11	26626	13.66	2.91
usb_funct	8114	25	29805	10.45	7416	25	28820	19.44	3.30
pci_bridge32	10336	34	40931	17.45	10365	37	39410	41.17	3.72
aes_core	10886	22	49225	20.23	11601	21	43725	30.18	11.17
wb_conmax	21656	17	100573	26.65	22541	15	95854	56.32	4.69
avg.				6.66				11.93	6.12

TABLE II
EXPERIMENTAL RESULTS OF TLN SYNTHESIS WITH $k = 15$.

circuit	LUT-based method [6]				Ours				R_Cost (%)
	#TLG	D	Cost	Time (s)	#TLG	D	Cost	Time (s)	
pci_conf.	56	3	261	0.04	52	3	255	0.07	2.30
steppermoto.	73	15	265	0.82	75	17	256	0.82	3.40
usb_phy	221	7	901	1.27	210	7	856	0.98	4.99
sasc	347	9	1588	1.33	360	9	1465	1.17	7.75
pci_spoci_ctrl	425	14	1534	12.37	321	15	1350	8.07	11.99
simple_spi	605	17	2010	3.95	447	16	1662	3.75	17.31
i2c	611	19	2034	5.68	473	18	1934	4.84	4.92
systemcdes	1273	20	5984	173.94	1423	20	5438	44.98	9.12
spi	1939	22	6690	131.06	1498	21	6417	28.34	4.08
des_area	2379	23	11517	213.27	2325	23	10837	66.71	5.90
tv80	3837	38	16231	193.35	3806	36	15188	68.70	6.43
mem_ctrl	4428	25	17543	95.69	4080	26	16957	65.81	3.34
systemcaes	4753	29	20874	225.06	4829	31	19226	96.11	7.89
ac97_ctrl	6863	8	27394	92.69	6388	11	26614	51.53	2.85
usb_funct	8092	25	29896	194.81	7369	25	28718	97.56	3.94
pci_bridge32	10335	34	41042	201.77	10149	37	38890	176.47	5.24
aes_core	11030	21	49316	462.64	11544	21	43471	181.23	11.85
wb_conmax	21068	18	97682	429.93	21890	15	93447	296.32	4.34
avg.				135.54				66.30	6.54

V. EXPERIMENTAL RESULTS

We implemented the proposed method in C within the ABC framework [7] and conducted experiments on a Linux system with an Intel Xeon Gold 5218 CPU and 256GB of memory. Benchmark circuits were selected from the IWLS 2005 suite [16]. For comparison, each benchmark was synthesized separately using the SOTA LUT-based method [6] and our method. Because our method requires more TF identifications with the extended DSD manager, technology mapping (via

TABLE III

COMPARISON OF EXPERIMENTAL RESULTS FOR 2-TLN SYNTHESIS WITH $k = 6$.

circuit	LUT-based method [6]				Our basic method				Our enhanced method				R_Cost (%)		
	#TLG	D	Cost	Time (s)	#TLG	D	Cost	Time (s)	#TLG	D	Cost	Time (s)	Bas. vs LUT	Enh. vs LUT	Enh. vs Bas.
pci_conf.	56	3	261	0.03	36	3	243	1.54	34	3	238	4.23	6.90	8.81	2.06
steppermotordrive	74	15	260	0.12	65	16	272	12.31	75	17	256	39.17	-4.62	1.54	5.88
usb_phy	221	7	903	0.21	147	6	1019	25.93	204	7	852	86.47	-12.85	5.65	16.39
sasc	347	9	1588	0.25	346	9	1550	19.12	333	9	1441	57.23	2.39	9.26	7.03
pci_spoci_ctrl	414	14	1498	0.83	332	15	1420	29.31	312	14	1346	135.01	5.21	10.15	5.21
simple_spi	605	17	2010	0.50	515	16	2051	23.81	427	16	1627	94.89	-2.04	19.05	20.67
i2c	611	19	2035	0.60	438	18	2082	28.42	448	18	1890	121.98	-2.31	7.13	9.22
systemcdes	1299	20	6220	3.39	1006	19	6407	230.41	1405	20	5407	608.27	-3.01	13.07	15.61
spi	1950	23	6701	2.99	1420	21	6621	61.77	1402	22	6289	285.57	1.19	6.15	5.01
des_area	2353	23	11326	5.95	2145	21	11297	235.78	2173	22	10967	640.17	0.26	3.17	2.92
tv80	3864	37	16250	9.93	3382	39	16560	140.74	3556	36	14942	552.78	-1.91	8.05	9.77
mem_ctrl	4483	26	17622	7.64	3782	27	17982	60.16	3833	26	16823	309.26	-2.04	4.53	6.45
systemcaes	4805	29	20691	7.73	4650	30	21103	142.88	4441	31	18792	468.68	-1.99	9.18	10.95
ac97_ctrl	6859	8	27425	4.87	5667	8	27415	43.56	5623	8	26164	182.38	0.04	4.60	4.56
usb_funct	8114	25	29805	10.45	7249	25	30329	99.49	7192	24	28378	499.54	-1.76	4.79	6.43
pci_bridge32	10336	34	40931	17.45	10128	36	40892	62.08	10169	38	39220	353.17	0.10	4.18	4.09
aes_core	10886	22	49225	20.23	9710	20	48642	212.61	11319	21	43437	822.13	1.18	11.76	10.70
wb_conmax	21656	17	100573	26.65	17656	15	93891	82.41	14464	16	89517	498.44	6.64	10.99	4.66
avg.				6.66				84.02				319.96	-0.48	7.89	8.20

the `&if` command in ABC) was performed only once per benchmark for efficiency. In contrast, the LUT-based method, by default, performed technology mapping three times per benchmark. All synthesized TLNs were verified using ABC's combinational equivalence checking command `cec`.

Table I presents the synthesis results using a cut size of 6 ($k = 6$), meaning that each TLG in the resulting TLNs has at most 6 fanins. Column 1 lists the benchmark circuits. Columns $2 \sim 5$ report the results of the LUT-based method, including the number of TLGs, logic depth, total area cost, and execution time. Columns $6 \sim 9$ show the corresponding results obtained using our method. Column 10 reports the percentage reduction in area cost achieved by our method compared to the LUT-based method.

The results demonstrate that our method is effective, achieving area reduction for all benchmark circuits, with an average improvement of 6.12%. Although our method incurs slightly higher execution time, it remains efficient, with all circuits synthesized in under one minute. Finally, the logic depth changes only slightly.

Table II presents the synthesis results with a cut size of 15 ($k = 15$). The results demonstrate that our method remains both effective and efficient for TLN synthesis at this larger cut size, achieving an average area reduction of 6.54%. Moreover, it is more efficient than the LUT-based method, as it requires only a single technology mapping while delivering better quality.

In addition, we adapted the LUT-based method for 2-TLN synthesis. The basic version replaces the original TF identification heuristic with ILP-based 2-TF identification [17] and performs technology mapping only once. Our enhanced version incorporates the two proposed improvements into the basic method, while also limiting technology mapping to a single pass.

Table III presents the experimental results with a cut size of 6. Columns $2 \sim 5$ report the results of the LUT-based method for conventional TLN synthesis. Columns $6 \sim 9$ show the results of our basic method for 2-TLN synthesis, while Columns $10 \sim 13$ present the results of our enhanced 2-TLN synthesis method. Columns $14 \sim 16$ show the area reductions achieved by: (1) our basic method compared to the LUT-based method, (2) our enhanced method compared to the LUT-based method, and (3) our enhanced method compared to our basic method.

The results show that the basic method does not consistently reduce area across all benchmarks compared to the LUT-based approach. This is somewhat unexpected, as 2-TF identification should theoretically produce equal or lower area costs per cut function than conventional TF identification. In contrast, the enhanced method consistently achieves area reduction. Therefore, simply replacing the TF identification heuristic in the LUT-based approach with the ILP-based 2-TF identification for 2-TLN synthesis is insufficient. Our enhancements for accurate area cost estimation are crucial for effective 2-TLN synthesis.

The overhead of the enhanced method arises from the increased execution time required for additional TF identifications with the extended DSD manager. However, this overhead is acceptable, as even the most time-consuming benchmark requires only about 800 seconds.

VI. CONCLUSION

This paper enhances the area estimation process in TLN synthesis by enhancing the DSD manager and selectively flipping cut functions. These improvements increase the accuracy of TLG area estimation while enabling effective function reuse. As a result, our method achieves greater area reduction than the SOTA LUT-based synthesis approach, with particularly significant gains in 2-TLN synthesis.

References

[1] R. O. Winder, "Single stage threshold logic," in *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pp. 321–332, 1961.

[2] R. Zhang, P. Gupta, L. Zhong, and N. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 107–118, 2005.

[3] A. K. Maan, D. A. Jayadevi, and A. P. James, "A survey of memristive threshold logic circuits," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 8, pp. 1734–1746, 2017.

[4] S.-Y. Lee, N.-Z. Lee, and J.-H. R. Jiang, "Searching parallel separating hyperplanes for effective compression of threshold logic networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[5] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski, "Identification of threshold functions and synthesis of threshold networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 665–677, 2011.

[6] A. Neutzling, J. M. Matos, A. Mishchenko, A. Reis, and R. P. Ribas, "Effective logic synthesis for threshold logic circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 926–937, 2019.

[7] Berkeley logic synthesis and verification group, "ABC: A system for sequential synthesis and verification, release 70930," 2007.

[8] A. Neutzling, M. G. A. Martins, V. Callegaro, A. I. Reis, and R. P. Ribas, "A simple and effective heuristic method for threshold logic identification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1023–1036, 2018.

[9] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for threshold logic circuit minimization," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014.

[10] A. K. Palaniswamy and S. Tragoudas, "Improved threshold logic synthesis using implicant-implicit algorithms," vol. 10, may 2014.

[11] H. Chen, S.-C. Hung, and J.-H. R. Jiang, "Disjoint-support decomposition and extraction for interconnect-driven threshold logic synthesis," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[12] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold logic circuits," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 396–403, 2011.

[13] N.-Z. Lee, H.-Y. Kuo, Y.-H. Lai, and J.-H. R. Jiang, "Analytic approaches to the collapse operation and equivalence verification of threshold logic circuits," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[14] A. Mishchenko and R. Brayton, "Faster logic manipulation for large designs," in *Proc. of Int'l Workshop on Logic and Synthesis*, 2007.

[15] S. Plaza and V. Bertacco, "Boolean operations on decomposed functions," *Proc. IWLS'05*, pp. 310–317, 2005.

[16] "IWLS 2005 benchmark suite," 2005.

[17] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, "A new method to identify threshold logic functions," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 934–937, 2017.

[18] 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 *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 469–474, 2021.

[19] Y.-S. Lin, Y.-C. Chen, L.-C. Zheng, and K.-C. Chen, "Enhanced 2nd-order threshold function identification with application to 2nd-order threshold logic network synthesis," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 31, Dec. 2025.

[20] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2284–2297, 2019.

[21] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to technology mapping for lut-based fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 240–253, 2007.

[22] A. Neutzling, J. M. Matos, A. I. Reis, R. P. Ribas, and A. Mishchenko, "Threshold logic synthesis based on cut pruning," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 494–499, 2015.

[23] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for lut-based fpgas," in *Proc. of Int'l Symp. on Field Programmable Gate Arrays*, 1998.

[24] Y.-C. Chen, R. Wang, and Y.-P. Chang, "Fast synthesis of threshold logic networks with optimization," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 486–491, 2016.

[25] Bertacco and Damiani, "The disjunctive decomposition of logic functions," in *1997 Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 78–82, 1997.

[26] S. Muroga, *Threshold logic and its applications*. New York, NY: John Wiley, 1971.