Constraint Solving for Synthesis and Verification of Threshold Logic Circuits

Nian-Ze Lee, *Student Member, IEEE*, and Jie-Hong R. Jiang, *Member, IEEE*

***Abstract*—Threshold logic (TL) circuits gain increasing attention due to their feasible realization with emerging technologies and strong bind to neural network applications. In this work, we devise techniques for automatic synthesis and verification of TL circuits based on constraint solving. For synthesis, we formulate a fundamental operation to collapse TL functions, and derive a necessary and sufficient condition of collapsibility for linear combination of two TL functions. An approach based on solving the subset sum problem is proposed for fast circuit transformation. For verification, we propose a procedure to convert a TL function to a multiplexer (MUX) tree and to pseudo-Boolean (PB) constraints for formal Boolean and PB reasoning, respectively. Experiments on synthesis show that the collapse operation further reduces gate counts of synthesized TL circuits by an average of 18%. Experiments on verification demonstrate good scalability of the MUX-based method for equivalence checking of synthesized TL circuits, and efficiency of PB constraint conversion in cases where the conjunctive normal form (CNF) formula conversion and MUX tree conversion suffer from memory explosion.**

***Index Terms*—Collapse operation, design automation, formal verification, logic synthesis, neural network, threshold logic circuit.**

I. Introduction

While the continuation of Moore's law slows down, different computation paradigms have been considered as possible substitutions for CMOS technologies. Among these possibilities, threshold logic (TL) circuits gain increasing attention due to their strong bind to neural network applications [5], [10], [11], [18], [27] and feasible realization with emerging technologies, such as spintronics, memristors, resonant tunneling devices, quantum cellular automata, and single electron transistors [1], [6], [23]. As the technologies realizing TL become more viable than before, the automatic synthesis and verification of TL circuits are important research topics to support large scale system construction. In this article, utilizing various kinds of constraint solving techniques, we devise effective and efficient approaches for both synthesis and verification of TL circuits.

For synthesis, a fundamental operation of collapsing (or composing) two TL functions is formulated, which subsumes the restricted merging operations in [3]. Building upon a necessary and sufficient condition for collapsibility of the linear combination of two TL functions, an approach based on solving the *subset sum* problem [4] is proposed for fast circuit transformation. Moreover, using the sufficient condition alone avoids solving the subset sum problem and still provides good approximation for collapsibility empirically, which further enhances the scalability of the collapse operation.

For verification, on top of the state-of-the-art approach [33], which translates a TL function into a conjunctive normal form (CNF) formula and relies on Boolean satisfiability (SAT) solvers for formal reasoning, we propose to convert a TL function into a multiplexer (MUX) tree, and apply the well-developed verification techniques for Boolean logic circuits, which not only rely on SAT solving but also achieve better scalability through structural analysis and circuit simulation. On the other hand, as the memory required to represent a TL function in terms of a CNF formula or a MUX tree in the worst case is exponential to the number of its input variables, i.e., *input size*, they might suffer from the memory explosion issue as the input size grows. To deal with this problem, we propose a linear-time translation from a TL function to pseudo Boolean (PB) constraints to facilitate verification with compact memory usage.

We remark that, collapsing TL functions increases the input size of the function and potentially results in functions with large input sizes, whose practicality might seem unclear. However, we mention a key application to justify the usefulness of collapse operation. In machine learning and brain emulation applications, a neuron in a neural network typically has a large input size. For example, a neuron of a deep convolutional neural network for image classification may have 2048 inputs [13]. It is therefore common for neuromorphic chips to support neurons with large input sizes, e.g., the IBM TrueNorth chip [19] implements programmable neurons with up to 256 inputs and may serve as a feasible platform to realize TL functions with large input sizes. As prior work on TL synthesis mostly considers TL functions with up to a few inputs, e.g., eight inputs in [22], the collapse operation may help relax the input-size restriction for TL circuits to be mapped into neuromorphic architectures. In addition, it may facilitate the integration of neural networks and control logic circuits into

Manuscript received July 28, 2019; revised February 14, 2020; accepted July 16, 2020. Date of publication August 10, 2020; date of current version April 21, 2021. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant 104-2628-E-002-013-MY3, Grant 105-2221-E-002-196-MY3, Grant 105-2923-E-002-016-MY3, and Grant 108-2221-E-002-144-MY3. A preliminary version of this manuscript is published in [16]. This article was recommended by Associate Editor P. Stanley-Marbell. *(Corresponding author: Jie-Hong R. Jiang.)*

Nian-Ze Lee is with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: d04943019@ntu.edu.tw).

Jie-Hong R. Jiang is with the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan, and also with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: jhjiang@ntu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2020.3015441

a single neuromorphic chip. On the other hand, the collapse (namely, composition) operation can be used as an elementary command, dual to the decomposition operation, in a synthesis script to transform TL circuits for optimization, similar to its Boolean logic counterpart used in conventional logic synthesis, such as command `eliminate` in `SIS` [28].

In our experiments, we synthesize Boolean logic circuits in `ISCAS` and `ITC` benchmark suites into TL circuits using the technique proposed in [22], and apply the collapse operation to further reduce TL gate (TLG) counts. The collapse operation achieves an average of 18% gate count reduction on top of synthesized TL circuits. We then evaluate the proposed MUX- and PB-based verification methods, using the equivalence checking between the TL circuits before and after collapsing as a case study. Compared to the CNF-based verification method, the PB-based one solves the same set of benchmark circuits, and the MUX-based one uniquely solves all benchmarks in a few minutes. To demonstrate the compact memory usage of the PB-based verification method, we study the translation of TL functions with large input sizes, and show that the PB-based verification method is applicable to benchmarks containing TL functions with up to 128 inputs, which arise from practical neural networks.

To sum up, the main results in this article include the following.

1) A formulation of the collapse operation for TL functions and a necessary and sufficient condition for collapsibility. The synthesis algorithm based on iteratively collapsing TLGs achieves an average of 18% gate count reduction on top of synthesized TL circuits.
2) A conversion from a TL function to a MUX tree, which enables the use of efficient verification techniques for Boolean logic circuits and uniquely solves the equivalence checking of TL circuits for all benchmark circuits.
3) A linear-time translation from a TL function to PB constraints, which is applicable to benchmarks containing TL functions with large input sizes arising from practical neural networks.

We emphasize that, unlike most previous synthesis and verification methods imposing restriction on the input size of a TL function, our proposed collapse operation and TL-to-PB translation relax the limitation on the input sizes. The relaxation on input sizes benefits the utilization of the proposed techniques in neural network applications, which typically involve TL functions with large input sizes.

The remainder of this article is organized as follows. Some of the related works are discussed in Section II. Preliminaries are given in Section III. The proposed collapse operation is formulated in Section IV. The proposed TL-to-MUX and TL-to-PB translation techniques are presented in Section V. Two applications, TL circuit collapsing and equivalence checking, are discussed in Section VI. Experimental results in Section VII show the effectiveness of collapse operation and efficient equivalence checking by the proposed MUX/PB-based verification methods. Section VIII concludes this article.

II. Related Work

In this section, we review some related works for the design automation of TL circuits, and compare our proposed techniques to relevant prior endeavors. We also discuss the connection between TL circuits and neural network applications.

A. Synthesis

Among prior synthesis endeavors, [32] splits a Boolean logic circuit into unate nodes and applies linear programming to derive corresponding weights and thresholds; [29] proposes a decomposition algorithm that works directly on truth tables and applies a binate splitting heuristic; a tree matching method is used in [7] to synthesize TL circuits; in [25], implicant-implicit algorithms are proposed to improve synthesis performance; [22] starts from the and-inverter-graph (AIG) representation of a Boolean logic circuit and utilizes the well-developed technology mapper in [34] to map an AIG with cuts that are TL functions. As [22] is one of the state-of-the-art synthesis approaches for TL circuits, we will employ it as a baseline in the empirical evaluation and investigate the room to further collapse TL functions in synthesized TL circuits.

B. Verification

For verification, previous work [8] translates a TL function into its maximally factored form and then converts it into a Boolean expression diagram (BED) [12]. The converted BED is then transformed to a binary decision diagram (BDD) for further formal reasoning. However, BDDs tend to suffer from the memory explosion issue when the input size of the TL function is large. Therefore, the method is not scalable. Prior work [33] applies a path search approach to enumerate the product terms for a sum-of-product (SOP) expression of a TL function. The SOP expression is then converted to a CNF formula for SAT-based reasoning. There are two drawbacks in the CNF-based approach. First, translating a TL circuit into a CNF formula and purely relying on SAT solving for verification is not efficient, as it overlooks structural information and circuit simulation to speedup the task. Second, it also suffers from the memory explosion issue: when the input size of a TL function is large, the CNF formula can blow up. In contrast, the proposed MUX-based verification method converts a TL circuit into a Boolean logic circuit, and hence enables the application of well-developed verification techniques, which involve structural comparison and random simulation to ease the burden of SAT solving and achieve better scalability. Moreover, the TL-to-PB translation converts a TL function to PB constraints in time linear to its input size, resulting in a compact representation of the TL function. As the CNF-based verification method in [33] is the state-of-the-art approach, we will compare the proposed MUX/PB-based techniques against it in the experiments.

C. Connection to Neural Network Applications

As neural networks obtain huge success in various tasks, including computer vision and natural language processing, many endeavors have been made to deploy them onto devices

with limited computational resource. Recently, taking advantage of binarized weights and activation functions, binarized neural networks (BNNs) have been proposed, and shown to achieve comparable performance to conventional neural networks using floating point computation [5], [11], [27]. In essence, a TL circuit is an activation-BNN, which only binarizes activation functions but not weights. Therefore, techniques developed for TL circuits are also applicable to BNNs and activation-BNNs. In our experiments, we conduct a case study over activation-BNNs and apply the proposed verification techniques to reason about their equivalence.

D. Other Topics

Beyond the synthesis and verification of TL, other topics, such as rewiring TL circuits [14], fast identification of TL functions [24], automatic test pattern generation [9], canonicalization [17], disjoint-support decomposition [2], and timing analysis [30], have also been investigated. In [3], the *merging* of TLGs is studied with a case-by-case manner. On the other hand, the formulated collapse operation is more general and hence subsumes the merging operation in [3]. In our empirical evaluation, the proposed collapse operation also achieves better optimization results compared to the statistics reported in [3].

III. Preliminaries

In this section, we introduce necessary background knowledge to facilitate subsequent discussions in this article. We denote Boolean constants $\bot$ and $\top$ by 0 and 1, respectively, and let $\mathbb{B} = \{0, 1\}$. Boolean connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are used under their conventional semantics. An n-variable Boolean function is a mapping from the n-dimensional Boolean space $\mathbb{B}^n$ to the 1-D Boolean space $\mathbb{B}$. Given an n-variable Boolean function $f(x_1, \ldots, x_n)$, its *positive cofactor* (resp. *negative cofactor*) with respect to variable x_i is an $(n-1)$-variable Boolean function f_{x_i} (resp. $f_{\neg x_i}$), which substitutes 1 (resp. 0) into variable x_i of function f and only refers to variables $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. The *Shannon expansion* of function f with respect to variable x_i equals $(x_i \wedge f_{x_i}) \vee (\neg x_i \wedge f_{\neg x_i})$.

A. Threshold Logic

A *TL function* $f : \mathbb{B}^n \to \mathbb{B}$ over Boolean variables $(x_1, \ldots, x_n)$ specified by a vector of constant weights $(w_1, \ldots, w_n) \in \mathbb{Z}^n$ and a constant threshold value $T \in \mathbb{Z}$ is a Boolean function such that

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

Note that the multiplication between Boolean variable x_i and its associated weight w_i is performed by treating Boolean constants 1 and 0 as integers.

A TLG v with n inputs $(x_1, \ldots, x_n)$ and one output z_v is a logical component realizing a TL function $f_v(x_1, \ldots, x_n)$ with weights $(w_1, \ldots, w_n)$ and a threshold value T_v. The valuation of its output z_v is determined by $f_v(x_1, \ldots, x_n)$. We denote a TLG v with weights $(w_1, \ldots, w_n)$ and a threshold value T_v as $v = [w_1, \ldots, w_n; T_v]$. We will not distinguish between a TL function and a TLG when it is clear from the context.

For a TLG $v = [w_1, \ldots, w_n; T_v]$ with inputs $(x_1, \ldots, x_n)$, we define the *positive cofactor* of v as a TLG

$$v_{x_i} = \left[w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n; T_v - w_i\right]$$

with inputs $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, which realizes the positive cofactor of the TL function $f_v(x_1, \ldots, x_n)$. Similarly, we define the *negative cofactor* of v as a TLG

$$v_{\neg x_i} = \left[w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n; T_v\right]$$

with inputs $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, which realizes the negative cofactor of the TL function $f_v(x_1, \ldots, x_n)$.

A *TL circuit* (abbreviated TL circuit, or TLC) $G = (V, E)$ is a directed acyclic graph (DAG), where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. An edge $e = (u, v)$ signifies that vertex v refers to vertex u as an input; u is called a *fanin* of v; v is called a *fanout* of u. The number of inputs of a vertex is called its *fanin size*. V_I (resp. V_O) is a nonempty subset of V such that every vertex in V_I (resp. V_O) has no fanins (resp. fanouts). We assume V_I and V_O are disjoint. A vertex $v \in V_I$ (resp. V_O) is referred to as a primary input (PI) [resp. primary output (PO)]. A vertex $v \in V \backslash V_I$ represents a TLG. In the sequel, we shall not distinguish between a vertex and its corresponding TLG.

IV. Collapse Operation of TL Circuits

In this section, we formulate the collapse operation for TLGs, and derive the necessary and sufficient conditions. The collapse operation can be seen as the opposite operation of decomposition, which decomposes a TLG into a composite of multiple TLGs.

A. Problem Formulation

First, we formulate the general collapsing as follows.

Problem Formulation 1 (General Collapsing): Given two TLGs $u = [a_1, \ldots, a_n; T_u]$ with inputs $(x_1, \ldots, x_n)$ and $v = [b_1, \ldots, b_m; T_v]$ with inputs $(y_1, \ldots, y_m)$, let u be a fanin of v, and assume $y_1 = z_u$. The *general TLG collapsing problem of u to v* asks whether there exists a TLG $\tilde{v} = [c_1, \ldots, c_{n+m-1}; T_{\tilde{v}}]$ with inputs $(x_1, \ldots, x_n, y_2, \ldots, y_m)$ such that $f_{\tilde{v}}(x_1, \ldots, x_n, y_2, \ldots, y_m) = f_v(f_u(x_1, \ldots, x_n), y_2, \ldots, y_m)$ for all truth assignments to variables $(x_1, \ldots, x_n, y_2, \ldots, y_m)$.

We remark that the general TLG collapsing problem formulated above involves $n+m$ parameters $c_1, \ldots, c_{n+m-1}, T_{\tilde{v}} \in \mathbb{Z}$ to search for a legitimate $\tilde{v}$. The large search space $\mathbb{Z}^{n+m}$ imposes expensive computation. To overcome this obstacle, we consider collapsing u to v in the special form of a linear combination of u and v as the following formulation states.

Problem Formulation 2 (Collapsing via Linear Combination): Given two TLGs $u = [a_1, \ldots, a_n; T_u]$ with inputs $(x_1, \ldots, x_n)$ and $v = [b_1, \ldots, b_m; T_v]$ with inputs $(y_1, \ldots, y_m)$, let u be a fanin of v, and assume $y_1 = z_u$. The *TLG collapsing problem of u to v via linear combination* asks whether there exists two positive parameters k and l such that the TLG $\tilde{v}(k, l) = [ka_1, \ldots, ka_n, lb_2, \ldots, lb_m; kT_u + l(T_v - b_1)]$ with inputs $(x_1, \ldots, x_n, y_2, \ldots, y_m)$ satisfies $f_{\tilde{v}}(x_1, \ldots, x_n, y_2,$

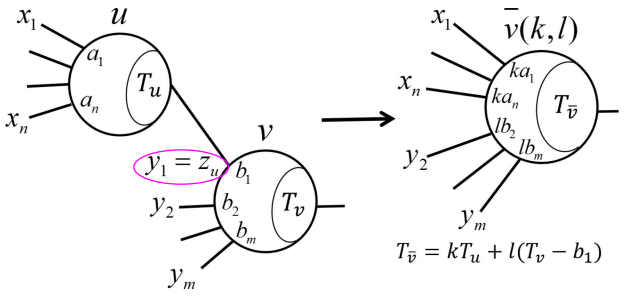


Fig. 1. Collapse of two TLGs via their linear combination.

$\ldots, y_m) = f_v(f_u(x_1, \ldots, x_n), y_2, \ldots, y_m)$ for all truth assignments to variables $(x_1, \ldots, x_n, y_2, \ldots, y_m)$.

Fig. 1 illustrates the above second formulation. Note that by collapsing u to v via their linear combination we can effectively reduce the dimensions of the parameter space from $n+m$ to 2. Below we discuss the feasibility conditions of the special collapsing under linear combination. We first assume $b_1 > 0$ and $\{x_1, \ldots, x_n\} \cap \{y_2, \ldots, y_m\} = \emptyset$ (i.e., inputs of u and v are disjoint). In Section IV-C, the cases of $b_1 < 0$ and $\{x_1, \ldots, x_n\} \cap \{y_2, \ldots, y_m\} \neq \emptyset$ (i.e., u and v have common inputs) will be discussed.

Example 1: As a concrete example of the problem formulation, consider TLGs $u = [4, 3; 5]$ with inputs (x_1, x_2) and $v = [2, 1; 3]$ with inputs (y_1, y_2), where $y_1 = z_u$. We ask whether there exist feasible parameters (k, l) to collapse u to v via their linear combination $\bar{v}(k, l) = [4k, 3k, l; 5k + l]$. The answer to this question will be obtained in Section IV-B.

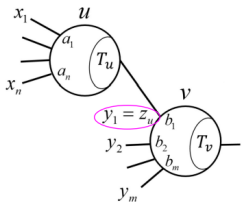
B. Collapsing Feasibility

To facilitate discussion, we define three functions

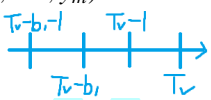
$$f_u^+(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \sum_{i=1}^{n} a_i x_i$$

$$f_v^-(y_2, \ldots, y_m) \stackrel{\text{def}}{=} \sum_{j=2}^{m} b_j y_j$$

$$f_{v \circ u} \stackrel{\text{def}}{=} f_v(f_u(x_1, \ldots, x_n), y_2, \ldots, y_m)$$



and four sets of truth assignments



$$\phi_1 \stackrel{\text{def}}{=} \{(\beta_2, \ldots, \beta_m) \mid f_v^-(\beta_2, \ldots, \beta_m) \leq T_v - b_1 - 1\}$$

$$\phi_2 \stackrel{\text{def}}{=} \{(\beta_2, \ldots, \beta_m) \mid f_v^-(\beta_2, \ldots, \beta_m) \geq T_v\}$$

$$\phi_3 \stackrel{\text{def}}{=} \{(\beta_2, \ldots, \beta_m) \mid T_v - b_1 \leq f_v^-(\beta_2, \ldots, \beta_m) \leq T_v - 1\}$$

$$\phi_4 \stackrel{\text{def}}{=} \{(\alpha_1, \ldots, \alpha_n) \mid f_u^+(\alpha_1, \ldots, \alpha_n) \leq T_u - 1\}$$

for $(\alpha_1, \ldots, \alpha_n) \in \mathbb{B}^n$ and $(\beta_2, \ldots, \beta_m) \in \mathbb{B}^{m-1}$.

Observe that any truth assignment in ϕ_1 makes $f_{v \circ u} = 0$, any truth assignment in ϕ_2 makes $f_{v \circ u} = 1$, and any truth assignment in ϕ_3 makes the valuation of $f_{v \circ u}$ purely determined by f_u. If $\phi_3 = \emptyset$, then $f_{v \circ u} = f_v$ for all truth assignments to variables $(x_1, \ldots, x_n, y_2, \ldots, y_m)$, i.e., the valuation of z_v does not depend on that of z_u. In the following derivation, this futile situation is excluded.

徒劳的

The necessary and sufficient conditions for collapsing u to v by a linear combination of weights are stated in Theorem 1.

Theorem 1: Given two TLGs $u = [a_1, \ldots, a_n; T_u]$ with inputs $(x_1, \ldots, x_n)$ and $v = [b_1, \ldots, b_m; T_v]$, $b_1 > 0$, with inputs $(y_1, \ldots, y_m)$, $y_1 = z_u$, and $\{x_1, \ldots, x_n\} \cap \{y_2, \ldots, y_m\} = \emptyset$, collapsing u to v via linear combination under parameter (k, l) yielding $\bar{v} = [ka_1, \ldots, ka_n, lb_2, \ldots, lb_m; kT_u + l(T_v - b_1)]$ such that $f_{\bar{v}} = f_{v \circ u}$ *if and only if* parameters (k, l) satisfy the following inequalities whose side conditions are met:

$$l(T_v - b_1 - \max_{\beta \in \phi_1}\{f_v^-\}) \geq k(\max\{f_u^+\} - T_u) + 1, \quad \phi_1 \neq \emptyset$$

$$l(\min_{\beta \in \phi_2}\{f_v^-\} + b_1 - T_v) \geq k(T_u - \min\{f_u^+\}), \quad \phi_2 \neq \emptyset$$

$$k(T_u - \max_{\alpha \in \phi_4}\{f_u^+\}) \geq l(\max_{\beta \in \phi_3}\{f_v^-\} + b_1 - T_v) + 1.$$

We remark that it is difficult to compute the coefficients in the inequalities of Theorem 1. For example, to compute $\max_{\beta \in \phi_1}\{f_v^-\}$, one needs to solve the *subset sum* problem, which is NP-complete [4]. This difficulty may be addressed by considering Theorem 2, which states a set of sufficient conditions and is much more efficient to compute.

Theorem 2: Given two TLGs $u = [a_1, \ldots, a_n; T_u]$ with inputs $(x_1, \ldots, x_n)$ and $v = [b_1, \ldots, b_m; T_v]$, $b_1 > 0$, with inputs $(y_1, \ldots, y_m)$, $y_1 = z_u$, and $\{x_1, \ldots, x_n\} \cap \{y_2, \ldots, y_m\} = \emptyset$, collapsing u to v via linear combination under parameter (k, l) yielding $\bar{v} = [ka_1, \ldots, ka_n, lb_2, \ldots, lb_m; kT_u + l(T_v - b_1)]$ such that $f_{\bar{v}} = f_{v \circ u}$ *if* parameters (k, l) satisfy the following inequalities whose side conditions are met:

$$l \geq k(\max\{f_u^+\} - T_u) + 1, \quad \phi_1 \neq \emptyset$$
$$lb_1 \geq k(T_u - \min\{f_u^+\}), \quad \phi_2 \neq \emptyset$$
$$k \geq l(b_1 - 1) + 1.$$

Notice that, although ϕ_1 and ϕ_2 are still involved as side conditions in Theorem 2, unlike those in Theorem 1, the max and min operations in Theorem 2 are not constrained by ϕ_1 and ϕ_2 and range over all truth assignments. Hence, the coefficients $\max\{f_u^+\}$ and $\min\{f_u^+\}$ in these inequalities of Theorem 2 can be computed in time linear to the fanin size of the TLG, which enhances computational efficiency of collapse operation. A tradeoff between the identification precision of TL functions and computational complexity is provided by Theorems 1 and 2.

Proof: To prove the above two theorems, we search for the conditions such that $f_{v \circ u}$ equals $f_{\bar{v}}$ under all truth assignments $(\alpha_1, \ldots, \alpha_n, \beta_2, \ldots, \beta_m)$ for variables $(x_1, \ldots, x_n, y_2, \ldots, y_m)$. There are three cases to discuss: First, $(\beta_2, \ldots, \beta_m) \in \phi_1$. Second, $(\beta_2, \ldots, \beta_m) \in \phi_2$. Third, $(\beta_2, \ldots, \beta_m) \in \phi_3$. Their corresponding derivations are obtained in Lemmas 1–3, respectively. ■

Lemma 1: For every truth assignment $(\alpha_1, \ldots, \alpha_n, \beta_2, \ldots, \beta_m)$ to $(x_1, \ldots, x_n, y_2, \ldots, y_m)$ such that $(\beta_2, \ldots, \beta_m) \in \phi_1$, i.e., $f_v^-(\beta_2, \ldots, \beta_m) \leq T_v - b_1 - 1$

1) iff-condition

$$f_{v \circ u} = f_{\bar{v}} \iff l(T_v - b_1 - \max_{\beta \in \phi_1}\{f_v^-\}) \geq k(\max\{f_u^+\} - T_u) + 1$$

2) if-condition

$$f_{vou} = f_{\bar{v}} \Longleftarrow l \geq k(\max\{f_u^+\} - T_u) + 1.$$

Proof: For any truth assignment in ϕ_1, we have $f_{vou} = 0$. In the following derivation, $f_u^+ > T_u$ is assumed, since if $f_u^+ \leq T_u$, $f_{\bar{v}} = f_{vou}$ holds trivially:

$$\begin{aligned} f_{\bar{v}} = 0 &\iff kf_u^+ + lf_v^- < kT_u + l(T_v - b_1) \\ &\iff k\left(f_u^+ - T_u\right) < l\left(T_v - b_1 - f_v^-\right) \\ &\iff \frac{k}{l} < \frac{T_v - b_1 - f_v^-}{f_u^+ - T_u} \\ &\iff \frac{k}{l} < \min\left\{\frac{T_v - b_1 - f_v^-}{f_u^+ - T_u}\right\} \\ &\iff \frac{k}{l} < \frac{T_v - b_1 - \max\limits_{\beta\in\phi_1}\{f_v^-\}}{\max\{f_u^+\} - T_u} \\ &\Longleftarrow \frac{k}{l} < \frac{1}{\max\{f_u^+\} - T_u}. \end{aligned}$$

From the above derivation, the lemma follows. ■

Example 2: Continue Example 1. As $\phi_1 \neq \emptyset$, the conditions in Lemma 1 should be satisfied by parameters (k, l). The conditions are

$$\begin{aligned} f_{vou} = f_{\bar{v}} &\iff l(3 - 2 - 0) \geq k(7 - 5) + 1 \\ f_{vou} = f_{\bar{v}} &\Longleftarrow l \geq k(7 - 5) + 1. \end{aligned}$$

Suppose the condition $l \geq 2k + 1$ is violated, e.g., by setting $l = 2k$. The resulting TLG $\bar{v}$ becomes [4, 3, 2; 7]. As predicted by Lemma 1, a discrepancy must occur. Indeed, we can find $(x_1, x_2, y_2) = (1, 1, 0)$ such that $f_{\bar{v}} = 1$ and $f_{vou} = 0$.

Lemma 2: For every truth assignment $(\alpha_1, \ldots, \alpha_n, \beta_2, \ldots, \beta_m)$ to $(x_1, \ldots, x_n, y_2, \ldots, y_m)$ such that $(\beta_2, \ldots, \beta_m) \in \phi_2$, i.e., $f_v^-(\beta_2, \ldots, \beta_m) \geq T_v$

1) iff-condition

$$\begin{aligned} &f_{vou} = f_{\bar{v}} \iff \\ &\quad l(\min_{\beta\in\phi_2}\{f_v^-\} + b_1 - T_v) \geq k(T_u - \min\{f_u^+\}) \end{aligned}$$

2) if-condition

$$f_{vou} = f_{\bar{v}} \Longleftarrow lb_1 \geq k(T_u - \min\{f_u^+\}).$$

Proof: For any truth assignment in ϕ_2, we have $f_{vou} = 1$. In the following derivation, $f_u^+ < T_u$ is assumed, since if $f_u^+ \geq T_u$, $f_{\bar{v}} = f_{vou}$ holds trivially:

$$\begin{aligned} f_{\bar{v}} = 1 &\iff kf_u^+ + lf_v^- \geq kT_u + l(T_u - b_1) \\ &\iff k\left(f_u^+ - T_u\right) \geq l\left(T_v - b_1 - f_v^-\right) \\ &\iff \frac{k}{l} \leq \frac{f_v^- + b_1 - T_v}{T_u - f_u^+} \\ &\iff \frac{k}{l} \leq \min\left\{\frac{f_v^- + b_1 - T_v}{T_u - f_u^+}\right\} \\ &\iff \frac{k}{l} \leq \frac{\min\limits_{\beta\in\phi_2}\{f_v^-\} + b_1 - T_v}{T_u - \min\{f_u^+\}} \\ &\Longleftarrow \frac{k}{l} \leq \frac{b_1}{T_u - \min\{f_u^+\}}. \end{aligned}$$

From the above derivation, the lemma follows. ■

Example 3: Continue Example 2. Since $\phi_2 = \emptyset$, the conditions in Lemma 2 need not be imposed for parameters (k, l).

Lemma 3: For every truth assignment $(\alpha_1, \ldots, \alpha_n, \beta_2, \ldots, \beta_m)$ to $(x_1, \ldots, x_n, y_2, \ldots, y_m)$ such that $(\beta_2, \ldots, \beta_m) \in \phi_3$, i.e., $T_v - b_1 \leq f_v^-(\beta_2, \ldots, \beta_m) \leq T_v - 1$

1) iff-condition

$$\begin{aligned} &f_{vou} = f_{\bar{v}} \iff \\ &\quad k(T_u - \max_{\alpha\in\phi_4}\{f_u^+\}) \geq l(\max_{\beta\in\phi_3}\{f_v^-\} + b_1 - T_v) + 1 \end{aligned}$$

2) if-condition

$$f_{vou} = f_{\bar{v}} \Longleftarrow k \geq l(b_1 - 1) + 1.$$

Proof: For any truth assignment in ϕ_3, we have $kf_u^+ + lf_v^- \geq kT_u + l(T_v - b_1) \iff f_u^+ \geq T_u$. In the following derivation, $f_u^+ < T_u$ is assumed, since if $f_u^+ \geq T_u$, the assumption $f_v^- \geq T_v - b_1$ implies $kf_u^+ + lf_v^- \geq kT_u + l(T_v - b_1)$, and $f_{\bar{v}} = f_{vou}$ holds trivially. Under the assumption of $f_u^+ < T_u$, $f_{vou} = 0$.

$$\begin{aligned} f_{\bar{v}} = 0 &\iff kf_u^+ + lf_v^- < kT_u + l(T_v - b_1) \\ &\iff k\left(f_u^+ - T_u\right) < l\left(T_v - b_1 - f_v^-\right) \\ &\iff \frac{k}{l} > \frac{f_v^- + b_1 - T_v}{T_u - f_u^+} \\ &\iff \frac{k}{l} > \max\left\{\frac{f_v^- + b_1 - T_v}{T_u - f_u^+}\right\} \\ &\iff \frac{k}{l} > \frac{\max\limits_{\beta\in\phi_3}\{f_v^-\} + b_1 - T_v}{T_u - \max\limits_{\alpha\in\phi_4}\{f_u^+\}} \\ &\Longleftarrow \frac{k}{l} > b_1 - 1 \end{aligned}$$

From the above derivation, the lemma follows. ■

Example 4: Continue Example 3. As $\phi_3 \neq \emptyset$, the conditions in Lemma 3 should be satisfied by parameters (k, l). The conditions are

$$\begin{aligned} f_{vou} = f_{\bar{v}} &\iff k(5 - 4) \geq l(1 + 2 - 3) + 1 \\ f_{vou} = f_{\bar{v}} &\Longleftarrow k \geq l(2 - 1) + 1. \end{aligned}$$

The condition $k \geq 1$ should not be violated since we consider $k > 0, l > 0$ in Problem Formulation 2. For the constraints derived from the iff-conditions (i.e., one from Lemma 1 and the other from Lemma 3) in Examples 2 to 4, they can be satisfied, e.g., by $(k, l) = (1, 3)$, which asserts that u can be collapsed to v yielding $\bar{v} = [4, 3, 3; 8]$. On the other hand, the derived constraints from if-conditions together yield no solution due to their under-approximation of (k, l) solutions.

We remark that the merging operations proposed in [3] are special cases of the collapse operation formulated above. For example, the AND *gate-based merging* in [3], which considers to "merge" an AND gate v with one of its fanins u, is subsumed by collapsing u to v in our formulation. Theorems 1 and 2 can be applied to derive feasible parameters to collapse u to v via their linear combination.

Due to the linearity of the inequality constraints, an advantage of the proposed collapse operation lies in that, after coefficients in the inequalities are computed, analytic solutions for parameters (k, l) can be obtained. We remark that, in

general, there may be multiple feasible solutions for parameters (k, l), and those with smaller magnitude for k and l are preferred since the collapsed TLG would have smaller weights and threshold values. A possible approach to derive feasible solutions analytically is to find the intersecting point of these constraints and rounding it up to integers. For example, suppose two inequalities $l \geq k(\max\{f_u^+\} - T_u) + 1$ and $k \geq l(b_1 - 1) + 1$ are derived based on the if-conditions, one can solve for the intersecting point of the two lines analytically. Let the coordinates of the intersecting point be (l^*, k^*). A feasible (k, l) combination can be derived by first rounding l^* up to $\lceil l^* \rceil$ and substituting $\lceil l^* \rceil$ into $k = l(b_1 - 1) + 1$ to obtain the corresponding k coordinate. Such analytic search results in a fast collapsing computation. Also, observe that in Examples 2 to 4, if u and v are canonicalized to $[1, 1; 2]$ before collapsing, both the iff-conditions and the if-conditions can yield the solution $(k, l) = (1, 1)$ to collapse u to v resulting in $\bar{v} = [1, 1, 1; 3]$. Therefore, applying canonicalization in [17] may enhance the feasibility of collapse operation.

C. Negative Weight and Nondisjoint Fanin

Theorems 1 and 2 assume $b_1 > 0$ and $\{x_1, \ldots, x_n\} \cap \{y_2, \ldots, y_m\} = \emptyset$. We show how to handle the cases when $b_1 < 0$ and $\{x_1, \ldots, x_n\} \cap \{y_2, \ldots, y_m\} \neq \emptyset$ by an example. The basic operations to complement an input of a TLG and to invert a TLG can be found in [21].

Example 5: Let $u = [1, 1; 2]$ with inputs (x_1, x_2) and $v = [-1, -1; 0]$ with inputs (y_1, y_2), where $y_1 = z_u$ and $y_2 = x_2$. We first make the negative weight positive by complementing y_1, resulting in $v^* = [1, -1; 1]$. The inverter generated by complementing y_1 is then combined with u, yielding $u^* = [-1, -1; -1]$. To collapse u to v, it is equivalent to collapse u^* to v^*, where the weight of y_1 is inverted to 1, and Theorem 1 can be applied by first assuming that x_2 and y_2 are different variables. In this example, one can verify that $(k, l) = (1, 2)$ is a feasible solution. As a result, the collapsed TLG is $\bar{v} = [-1, -1, -2; -1]$ with inputs (x_1, x_2, y_2). Since x_2 and y_2 are actually the same variable, their weights should be summed up and the resulting TLG is $\bar{v} = [-1, -3; -1]$ with inputs (x_1, x_2).

V. Verification of TL Circuits

In this section, we propose two novel approaches for the verification of TL circuits. The first approach modifies the algorithm *path search with weight ordering* proposed in [33] and converts a TLG into a *multiplexer tree* (MUX tree), i.e., a tree-shaped Boolean logic circuit whose logic gates are MUXes. As a result, the efficient verification techniques for Boolean logic circuits, e.g., command `cec` in ABC [34] for *combinational equivalence checking*, can be applied. Second, we propose to translate a TL function into two PB constraints, in time linear to the input size of the TL function, and solve the derived PB constraints for verification. As to be shown in the experimental evaluation, the MUX-based verification method is more efficient for benchmarks synthesized from conventional circuits, where the fanin sizes of TLGs are small (roughly less than 20), and the PB-based verification method is applicable to benchmarks with fanin size up to 128, arising from neural network applications.

ConvertTLGMUX
input: a TLG v and a Boolean logic circuit C
output: a Boolean logic circuit C' equivalent to v
begin
01 **if** *IsConstOne*(v)
02 **return** `CONST1`;
03 **if** *IsConstZero*(v)
04 **return** `CONST0`;
05 x_{i^*} := *SelectMaxAbsWeigthInput*(v);
06 m := *CreateMuxGate*(C);
07 m.*SetControlInput*(x_{i^*});
08 m.*SetDataZeroInput*(*ConvertTLGMux*($v_{\neg x_{i^*}}$, C).`PO`);
09 m.*SetDataOneInput*(*ConvertTLGMux*($v_{x_{i^*}}$, C).`PO`);
10 C.*SetPrimaryOutput*(m);
11 **return** C;
end

Fig. 2. Algorithm: Convert a TLG into a MUX tree.

A. Conversion to MUX-Tree Boolean Logic Circuits

The *path search with weight ordering* algorithm in [33], which applies recursive Shannon expansion to a given TLG, can be modified to construct a MUX tree as follows. During the process of path search of a given TLG, instead of generating a clause for each path leading to 1 or 0, a MUX is created to simulate the Shannon expansion operation. The pseudo code of the proposed TL-to-MUX construction is outlined in Fig. 2. The inputs to the algorithm *ConvertTLGMUX* are a TLG v and a Boolean logic circuit whose PIs and output are the inputs and output of v, respectively, and has no logic gate initially. Lines `01-04` handle the base case to check whether TLG v is constant. If v always valuates to 1 (resp. 0), the gate `CONST1` (resp. `CONST0`), which always outputs 1 (resp. 0), is returned. If v is not constant, in line `05` the *weight ordering* heuristic proposed in [33] is applied to select the input x_{i^*} whose absolute value of the weight is the maximum among all inputs. Shannon expansion is then performed with respect to x_{i^*}, which is simulated by creating a MUX m in line `06`, whose controlling input is set to x_{i^*}. The data-zero input (resp. data-one input) of m is set to the PO of the circuit obtained by applying algorithm *ConvertTLGMUX* recursively to TLG $v_{\neg x_{i^*}}$ (resp. $v_{x_{i^*}}$). Finally, the output of m is connected to the PO of Boolean logic circuit C. Given a TL circuit, following a topological order from PIs to POs, we apply algorithm *ConvertTLGMUX* to every TLG inside it, and obtain an equivalent Boolean logic circuit. As a result, the verification of the TL circuit is reduced to that of the Boolean logic circuit, which involves structural analysis, circuit simulation, and SAT solving to improve scalability.

1) Redundancy Detection: Given a Boolean logic function $f(x_1, \ldots, x_n)$, variable x_i is *irredundant* in function f if the positive and negative cofactors of function f with respect to variable x_i are not equivalent, i.e., $f_{x_i} \neq f_{\neg x_i}$ for some truth assignment to variables $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Similarly, given a TLG v, an input x of v is irredundant in v if variable x is irredundant in function f_v.

The variable selection heuristic used in algorithm *ConvertTLGMUX* has a property that, every selected variable in line 05 for Shannon expansion is irredundant in the TLG v, which follows from the theorem below.

Theorem 3: Consider a TLG $v = [a_1, \ldots, a_n; T]$ with inputs $(x_1, \ldots, x_n)$. Assume v is not constant, i.e., neither v always valuates to 1 nor v always valuates to 0, and let $i^* = \arg\max_{i \in \{1,\ldots,n\}} |a_i|$. Then variable x_{i^*} is irredundant in TLG v.

Proof: Without loss of generality, assume $a_{i^*} > 0$. The case for $a_{i^*} < 0$ can be proved in a similar way. As TLG v is not constant, there exist two truth assignments $\tilde{z_1}$ and $\tilde{z_2}$ to variables $(x_1, \ldots, x_n)$ such that $f_v(\tilde{z_1}) \neq f_v(\tilde{z_2})$ and the Hamming distance between $\tilde{z_1}$ and $\tilde{z_2}$ equals 1. If $\tilde{z_1}$ and $\tilde{z_2}$ happen to differ in the valuation of variable x_{i^*}, then variable x_i^* is irredundant in TLG v because the valuation of variables $(x_1, \ldots, x_{i^*-1}, x_{i^*+1}, \ldots, x_n)$ in $\tilde{z_1}$ and $\tilde{z_2}$ witnesses the fact that $f_{v_{x_i}} \neq f_{v_{\neg x_i}}$.

On the other hand, suppose $\tilde{z_1}$ and $\tilde{z_2}$ differ in the valuation of variable x_j, and $j \neq i^*$. Without loss of generality, assume $a_j > 0$. The case for $a_j < 0$ can be proved in a similar way. For simplicity, assume $f_v(\tilde{z_1}) = 0$ and $f_v(\tilde{z_2}) = 1$, which implies $x_j = 0$ in $\tilde{z_1}$ and $x_j = 1$ in $\tilde{z_2}$. Depending on the valuation of variable x_{i^*} in $\tilde{z_1}$ and $\tilde{z_2}$, there are two cases.

First, if $x_{i^*} = 0$ in both $\tilde{z_1}$ and $\tilde{z_2}$, consider a truth assignment $\tilde{z_3}$ to variables $(x_1, \ldots, x_n)$ which is identical to $\tilde{z_1}$ except that in $\tilde{z_3}$, $x_{i^*} = 1$. As $a_{i^*} \geq a_j$, we have $f_v(\tilde{z_3}) = 1$. Note that $\tilde{z_1}$ and $\tilde{z_3}$ only differ in the valuation of variable x_{i^*}, and $f_v(\tilde{z_1}) \neq f_v(\tilde{z_3})$. Therefore, the valuation of variables $(x_1, \ldots, x_{i^*-1}, x_{i^*+1}, \ldots, x_n)$ in $\tilde{z_1}$ and $\tilde{z_3}$ witnesses the fact that x_{i^*} is irredundant in TLG v.

Second, if $x_{i^*} = 1$ in both $\tilde{z_1}$ and $\tilde{z_2}$, consider a truth assignment $\tilde{z_4}$ to variables $(x_1, \ldots, x_n)$ which is identical to $\tilde{z_2}$ except that in $\tilde{z_2}$, $x_{i^*} = 0$. As $a_{i^*} \geq a_j$, we have $f_v(\tilde{z_4}) = 0$. Note that $\tilde{z_2}$ and $\tilde{z_4}$ only differ in the valuation of variable x_{i^*}, and $f_v(\tilde{z_2}) \neq f_v(\tilde{z_4})$. Therefore, the valuation of variables $(x_1, \ldots, x_{i^*-1}, x_{i^*+1}, \ldots, x_n)$ in $\tilde{z_2}$ and $\tilde{z_4}$ witnesses the fact that x_{i^*} is irredundant in TLG v.

With the cases analyzed above, the theorem follows. ■

In algorithm *ConvertTLGMUX*, given a TLG v, lines 01–04 check whether v is constant; if v is not constant, line 05 selects the input x_{i^*} whose absolute value of the weight $|a_{i^*}|$ is the maximum among all inputs. According to Theorem 3, variable x_{i^*} is irredundant in TLG v.

2) Implementation Details: We remark that, in our implementation of algorithm *ConvertTLGMUX*, we use the AIG package in [34] as the underlying circuit representation for a MUX tree. Due to the structural hashing property [20] of the AIG package, automatic logic sharing between identical structures occurs during the construction of a MUX tree, compressing the MUX tree into a DAG. The logic sharing helps save the memory consumption of the construction, compared to the original conversion to CNF formulas in [33].

B. Translation to Pseudo Boolean Constraints

As an analogy to Tseitin's conversion translating Boolean logic circuits to CNF formulas [31], we propose a linear-time translation from TL circuits to PB constraints. Our translation converts a TL function to exactly two PB constraints, both of which are of length linear to the input size of the TL function. The conjunction of all the translated PB constraints yields the consistency condition of variable assignments to the entire TL circuit. Our translation facilitates formal reasoning of TL circuits with PB constraint solving, similar to reasoning of Boolean logic circuits with SAT solving. The TL-to-PB translation is detailed as follows.

Given a TLG $u = [a_1, \ldots, a_n; T]$ with inputs $(x_1, \ldots, x_n)$ and output y, we would like to derive PB constraints stating the relation $y \leftrightarrow (\sum_{i=1}^n a_i x_i \geq T)$, which is unfortunately not in the right format of PB constraints. The two directions of implication, $y \rightarrow (\sum_{i=1}^n a_i x_i \geq T)$ and $y \leftarrow (\sum_{i=1}^n a_i x_i \geq T)$, are translated into one PB constraint, respectively, as shown in the following lemmas.

Lemma 4: Given a TLG $u = [a_1, \ldots, a_n; T]$ with inputs $(x_1, \ldots, x_n)$ and output y, a truth assignment to variables $(x_1, \ldots, x_n, y)$ satisfies $y \rightarrow \sum_{i=1}^n a_i x_i \geq T$ if and only if it satisfies the following PB constraint:

$$\left(\sum_{i=1}^{n} a_i x_i - T\right) + (-m + T)(1 - y) \geq 0 \qquad (1)$$

where m is the summation of all negative weights among $\{a_1, \ldots, a_n\}$. It is defined to be 0 if every weight $a_i \geq 0$.

Proof: Let $A \subseteq \mathbb{B}^{n+1}$ be the set of truth assignments to variables $(x_1, \ldots, x_n, y)$ that satisfy $y \rightarrow \sum_{i=1}^n a_i x_i \geq T$; let $B \subseteq \mathbb{B}^{n+1}$ be the set of truth assignments to variables $(x_1, \ldots, x_n, y)$ that satisfy (1). We show $A = B$ by establishing $A \subseteq B$ and $B \subseteq A$ as follows.

To show that $A \subseteq B$, consider an arbitrary truth assignment $\tilde{z} = (\alpha_1, \ldots, \alpha_n, \beta) \in A$. There are two cases: when $\beta = 1$ and $\sum_{i=1}^n a_i \alpha_i \geq T$, (1) is satisfied by $\tilde{z}$ because by substituting $\tilde{z}$ into (1) we have $\sum_{i=1}^n a_i \alpha_i \geq T$, which is our assumption. On the other hand, when $\beta = 0$, there is no restriction for $(\alpha_1, \ldots, \alpha_n)$ according to $y \rightarrow \sum_{i=1}^n a_i x_i \geq T$. Indeed, substituting $\tilde{z}$ into (1), we have $\sum_{i=1}^n a_i \alpha_i - m \geq 0$, which holds for any $(\alpha_1, \ldots, \alpha_n)$ since m is defined to be the summation of all negative a_i. From the above two cases, we have shown that $\tilde{z} \in B$, and thus $A \subseteq B$.

To show that $B \subseteq A$, consider an arbitrary truth assignment $\tilde{z} = (\alpha_1, \ldots, \alpha_n, \beta) \in B$. There are two cases: When $\beta = 1$, (1) implies $\sum_{i=1}^n a_i \alpha_i \geq T$. Therefore, $\tilde{z}$ satisfies $y \rightarrow \sum_{i=1}^n a_i x_i \geq T$. On the other hand, when $\beta = 0$, $\tilde{z}$ also satisfies $y \rightarrow \sum_{i=1}^n a_i x_i \geq T$. From the above two cases, we have shown that $\tilde{z} \in A$, and thus $B \subseteq A$.

With the cases analyzed above, the lemma follows. ■

Lemma 5: Given a TLG $u = [a_1, \ldots, a_n; T]$ with inputs $(x_1, \ldots, x_n)$ and output y, a truth assignment to variables $(x_1, \ldots, x_n, y)$ satisfies $y \leftarrow \sum_{i=1}^n a_i x_i \geq T$ if and only if it satisfies the following PB constraint:

$$(T - 1 - \sum_{i=1}^{n} a_i x_i) + (-T + 1 + M)y \geq 0 \qquad (2)$$

where M is the summation of all positive weights among $\{a_1, \ldots, a_n\}$. It is defined to be 0 if every weight $a_i \leq 0$.

Proof: Let $A \subseteq \mathbb{B}^{n+1}$ be the set of truth assignments to variables $(x_1, \ldots, x_n, y)$ that satisfy $y \leftarrow \sum_{i=1}^n a_i x_i \geq T$; let $B \subseteq \mathbb{B}^{n+1}$ be the set of truth assignments to variables

$(x_1, \ldots, x_n, y)$ that satisfy (2). We show $A = B$ by establishing $A \subseteq B$ and $B \subseteq A$ as follows.

To show that $A \subseteq B$, consider an arbitrary truth assignment $\tilde{z} = (\alpha_1, \ldots, \alpha_n, \beta) \in A$. There are two cases: When $\beta = 0$ and $\sum_{i=1}^{n} a_i\alpha_i \leq T - 1$, (2) is satisfied by $\tilde{z}$ because by substituting $\tilde{z}$ into (2) we have $\sum_{i=1}^{n} a_i\alpha_i \leq T - 1$, which is our assumption. On the other hand, when $\beta = 1$, there is no restriction for $(\alpha_1, \ldots, \alpha_n)$ according to $y \leftarrow \sum_{i=1}^{n} a_i x_i \geq T$. Indeed, substituting $\tilde{z}$ into (2), we have $M - \sum_{i=1}^{n} a_i\alpha_i \geq 0$, which holds for any $(\alpha_1, \ldots, \alpha_n)$ since M is defined to be the summation of all positive a_i. From the above two cases, we have shown that $\tilde{z} \in B$, and thus $A \subseteq B$.

To show that $B \subseteq A$, consider an arbitrary truth assignment $\tilde{z} = (\alpha_1, \ldots, \alpha_n, \beta) \in B$. There are two cases: When $\beta = 0$, (1) implies $\sum_{i=1}^{n} a_i\alpha_i \leq T - 1$. Therefore, $\tilde{z}$ satisfies $y \leftarrow \sum_{i=1}^{n} a_i x_i \geq T$. On the other hand, when $\beta = 1$, $\tilde{z}$ also satisfies $y \leftarrow \sum_{i=1}^{n} a_i x_i \geq T$. From the above two cases, we have shown that $\tilde{z} \in A$, and thus $B \subseteq A$.

With the cases analyzed above, the lemma follows. ■

Combining Lemmas 4 and 5, we have the following theorem.

Theorem 4: Given a TLG $u = [a_1, \ldots, a_n; T]$ with inputs $(x_1, \ldots, x_n)$ and output y, a truth assignment to variables $(x_1, \ldots, x_n, y)$ satisfies $y \leftrightarrow \sum_{i=1}^{n} a_i x_i \geq T$ if and only if it satisfies the following two PB constraints:

$$\left(\sum_{i=1}^{n} a_i x_i - T\right) + (-m + T)(1 - y) \geq 0$$
$$\left(T - 1 - \sum_{i=1}^{n} a_i x_i\right) + (-T + 1 + M)y \geq 0$$

where m and M are defined in Lemmas 4 and 5, respectively.

Note that the length of both PB constraints above is linear to the fanin size of the given TLG. Below we use an example to illustrate how to translate a TLG into two PB constraints according to Theorem 4.

Example 6: Consider a TLG $u = [1, 1, 2; 2]$ with inputs (x_1, x_2, x_3) and output y. TLG u is translated into two PB constraints

$$(x_1 + x_2 + 2x_3 - 2) + 2(1 - y) \geq 0$$
$$(1 - x_1 - x_2 - 2x_3) + 3y \geq 0$$

according to Theorem 4.

Given a TL circuit, we apply Theorem 4 to translate every TLG inside it into PB constraints, and the conjunction of all PB constraints captures the consistent valuation of variables in the original TL circuit. Therefore, the formal reasoning of the TL circuit is reduced to solving PB constraints.

1) Plaisted-Greenbaum Encoding: When translating a TL circuit into its corresponding PB constraints, instead of establishing a bidirectional implication for every TLG with Theorem 4, the *polarity* of a TLG can be taken into account to establish a unidirectional implication using either Lemmas 4 and 5. This polarity-aware translation scheme is known as Plaisted-Greenbaum encoding (PG encoding) [26].

Consider a TL circuit G and a TLG $u = [a_1, \ldots, a_n; T]$ with inputs $(x_1, \ldots, x_n)$ and output y. TLG u is said to occur *positively* in G if for every path from u to any PO of G, the number of negative weights is even, and *negatively* in G if the number is odd. To translate G into PB constraints using Plaisted-Greenbaum encoding, every TLG occurring positively (resp. negatively) in G is converted to one PB constraint using (1) [resp. (2)] in Lemma 4 (resp. Lemma 5), establishing a unidirectional implication $y \rightarrow \sum_{i=1}^{n} a_i x_i \geq T$ (resp. $y \leftarrow \sum_{i=1}^{n} a_i x_i \geq T$). For TLGs occurring neither positively nor negatively, they are converted to two PB constraints using Theorem 4 to establish their bidirectional implications. Clearly, the number of translated PB constraints can be reduced using Plaisted-Greenbaum encoding. We will assess this effect empirically in our experiments.

```
CollapseNtk
  input: a TL circuit G = (V, E) and a bound B
  output: a collapsed TL circuit G' = (V', E')
  begin
  01   unmark every v ∈ V;
  02   while some v ∈ V is unmarked
  03     foreach v ∈ V
  04       foreach fanin u of v
  05         if |fanouts(u)| ≤ B
  06           if u can be collapsed to all of its fanouts
  07             foreach fanout t of u
  08               w := CollapseNode(u,t);
  09               unmark w;
  10               V := V\{t} ∪ {w};
  11             V := V\{u};
  12             continue; //to next v
  13         if u is the last fanin of v //no collapse
  14           mark v;
  15   return (V, E);
  end
```

Fig. 3. Algorithm: Collapse a TL circuit.

VI. Applications

In this section, we apply the proposed techniques to the automatic synthesis and verification of TL circuits. Specifically, the collapse operation is applied in the synthesis of TL circuits to reduce the TLG counts. On the other hand, the proposed TL-to-MUX and TL-to-PB translations are applied to verify the equivalence of two TL circuits.

A. Synthesis

Given a TL circuit, we apply the collapse operation repeatedly to reduce its TLG count. A TLG can be eliminated from the circuit if it can be collapsed into all of its fanouts. We remark that reducing the TLG count of a TL circuit may not be the only optimization objective. There can be other cost metrics in terms of, e.g., the fanin size, the weight values, and the threshold value of a TLG. If large fanin sizes or large weight/threshold values are not desirable, additional constraints can be imposed to restrict the collapse operation.

Fig. 3 sketches the procedure *CollapseNtk*, which reduces the TLG count of the given TL circuit by repeatedly applying the collapse operation. It iterates over every TLG v, and for each fanin u of v, a collapsibility check (line 06) is applied on u and returns true if and only if u can be collapsed to all

of its fanouts. The procedure then collapses u into its fanouts (lines07–10) if the collapsibility check suceeds. TLG v is marked if u is the last fanin of it (line 13), i.e., none of v's fanins can be collapsed into it; *CollapseNtk* terminates when all of the TLGs are marked. As a side remark, we also experimented with another collapsing procedure which applies the collapsibility check directly on v, and marks v when it cannot be collapsed to its fanouts. Our evaluation showed that this procedure did not achieve better quality of results. A possible explanation is the difference in the marking criterion. In *CollapseNtk*, TLG v is marked if "all" of its fanins cannot be collapsed into it; however, the second procedure marks v if it cannot be collapsed to "one" of its fanouts. Therefore, *CollapseNtk* marks TLGs less eagerly and is able to explore more optimization opportunities.

In our implementation, we resort to the sufficient conditions stated in Theorem 2 to compute the collapse parameters (k, l) due to its low computation cost. Our empirical experience suggests that the iff-conditions in Theorem 1 offer negligible improvement in TLG count reduction, and thus the if-conditions already provide good approximation to the sufficient and necessary criteria for collapse.

CollapseNtk has an additional parameter B that constrains the fanout size of a gate when it is considered to be collapsed into its fanouts, in line 05 of Fig. 3. To explore the effect of parameter B, we devise an iterative collapsing strategy, which executes *CollapseNtk* multiple times with increasing values for B. As to be seen in the experiments, compared to collapsing without limitation on the fanout size, i.e., running *CollapseNtk* with $B = \infty$ directly, the iterative strategy has better performance in terms of minimizing TLG counts. This phenomenon can be explained by the fact that if no limitation is imposed from the beginning, a TLG with large fanout size would be collapsed to all its fanouts earlier if collapsible, resulting in a large number of collapsed TLGs, which are more difficult to be collapsed with other TLGs. Instead, if we set a bound to the fanout size and iteratively increases the bound, we can mitigate the increasing noncollapsibility.

B. Verification

Given two TL circuits, we aim at verifying whether they have the same functionality, i.e., producing the same output response under identical input stimulus. The task is commonly known as *equivalence checking*. It is often applied over a circuit and its synthesized version to check if the functionality of the circuit remains intact after the synthesis procedure.

1) MUX-Based Verification Method: Given two TL circuits, we apply the proposed TL-to-MUX conversion to transform them into two equivalent Boolean logic circuits, respectively, and apply the well-developed equivalence checking techniques for Boolean logic circuits, such as command `cec` in [34].

2) PB-Based Verification Method: Given two TL circuits, we apply the proposed TL-to-PB conversion for their equivalence checking. A miter is built to assert the equivalence relation between two TL circuits by disjoining the XORs of corresponding output pairs. For each TLG in the miter, Theorem 4 is applied to translate it into PB constraints. For the added output equivalence circuitry, it can be first translated by Tseitin conversion into a CNF formula and further translated to PB constraints. After all the PB constraints are obtained, we set the objective function of the PB constraint solver to maximize the value of the miter output variable. The maximum value equals 0 if and only if the two TL circuits under verification are equivalent.

VII. Experimental Results

The collapsing-based synthesis and MUX/PB-based equivalence verification algorithms for TL circuits discussed in Section VI were implemented in the `ABC` environment [34]; the combinational logic of circuits in `ISCAS` and `ITC` benchmark suites were used for experiments. The implementation, benchmarks, and experimental data of this work are made publicly available.[1] We remark that studying the synthesis and verification of TL circuits with conventional circuit benchmarks could benefit the integration of control logic into neuromorphic computing architecture. We refer to the architecture parameters of TrueNorth chip [19] and impose a fanin size limitation of 256 and a weight/threshold value limitation in the range of $[-255, 255]$ for a TLG.

All experiments were conducted on a machine with Intel Xeon E5-2620 v4 2.1 GHz CPU, 126 GB RAM, and operating system CentOS 7.3.1611 (64 bit). The implementation was successfully compiled with compiler `gcc-8.2.0`. For each execution, we imposed a CPU time limit of two h and a memory usage limit of 1 GB. We use `TO` and `MO` to denote time-out and memory-out, respectively, in the tables below.

The following experiments were conducted. For synthesis, the collapsing-based technique is applied to both AIG circuits and synthesized TL circuits. As discussed in Section VI-A, two collapse strategies were applied: one with fanout bound parameter B set to infinity directly and the other incrementing the bound from 1 up to 100 iteratively. For verification, we first study the equivalence checking between the pairs of TL circuits before and after collapsing. The effect of Plaisted-Greenbaum encoding discussed in Section V-B1 was also assessed. Second, we evaluated the TLG translation scalability with respect to fanin size, and performed a case study over benchmarks derived from activation-BNNs, which contain TLGs with up to 128 inputs.

A. Synthesis

1) Collapsing AIG Circuits: The proposed collapse operation can be directly applied to AIG circuits as an AIG node can be easily translated to a TLG. For example, an AND gate is translated to a TLG $[1, 1; 2]$ and a NAND gate is translated to a TLG $[-1, -1; -1]$. We experimented on collapsing a *functionally reduced* AIG [20] (synthesized by command `fraig` in ABC) into a TL circuit, and the results are shown in Table I. Columns 4 and 5 report the numbers of AIG nodes and logic levels of the AIG circuits, respectively; Columns 6, 7, and 8 report the number of TLGs, the number of logic levels of the TL circuits, and the CPU time, respectively, for direct collapsing; Columns 9, 10, and 11 report the number of TLGs,

[1] https://github.com/NTU-ALComLab/TLCollapseVerify

TABLE I
STATISTICS OF COLLAPSING AIG CIRCUITS

benchmarks profile			statistics of AIG		statistics of collapsed TLC (w/o ite)			statistics of collapsed TLC (w/ ite)		
circuit	#pi	#po	#AIG	#level	#TLG	#level	time (s)	#TLG	#level	time (s)
c6288	32	32	2334 (1.00)	120 (1.00)	1407 (0.60)	93 (0.78)	0.01	1404 (0.60)	95 (0.79)	0.12
c7552	207	108	1961 (1.00)	29 (1.00)	991 (0.51)	22 (0.76)	0.01	846 (0.43)	19 (0.66)	0.07
s13207	31	121	2605 (1.00)	33 (1.00)	1213 (0.47)	21 (0.64)	0.01	1190 (0.46)	20 (0.61)	0.06
s15850	14	87	3330 (1.00)	46 (1.00)	1601 (0.48)	34 (0.74)	0.01	1479 (0.44)	32 (0.70)	0.07
s35932	35	320	10124 (1.00)	14 (1.00)	5078 (0.50)	9 (0.64)	0.02	4758 (0.47)	8 (0.57)	0.13
s38417	28	106	9062 (1.00)	30 (1.00)	4747 (0.52)	21 (0.70)	0.02	4388 (0.48)	19 (0.63)	0.25
s38584	12	278	11646 (1.00)	34 (1.00)	4981 (0.43)	22 (0.65)	0.04	4639 (0.40)	20 (0.59)	0.23
b14	32	54	5609 (1.00)	65 (1.00)	2866 (0.51)	49 (0.75)	0.02	2565 (0.46)	53 (0.82)	0.23
b15	36	70	8158 (1.00)	65 (1.00)	4028 (0.49)	47 (0.72)	0.03	3667 (0.45)	44 (0.68)	0.24
b17	37	97	26389 (1.00)	93 (1.00)	13379 (0.51)	66 (0.71)	0.13	12027 (0.46)	73 (0.78)	0.83
b18	37	23	77757 (1.00)	132 (1.00)	39733 (0.51)	98 (0.74)	0.31	35343 (0.45)	87 (0.66)	4.58
b19	24	27	156224 (1.00)	136 (1.00)	80621 (0.52)	105 (0.77)	0.75	70765 (0.45)	85 (0.63)	7.21
b20	32	22	11552 (1.00)	66 (1.00)	5929 (0.51)	47 (0.71)	0.05	5284 (0.46)	58 (0.88)	0.48
b21	32	22	11728 (1.00)	70 (1.00)	6017 (0.51)	54 (0.77)	0.05	5381 (0.46)	59 (0.84)	0.43
b22	32	22	17614 (1.00)	68 (1.00)	9071 (0.51)	57 (0.84)	0.10	8028 (0.46)	60 (0.88)	0.61
geomean			(1.00)	(1.00)	(0.50)	(0.73)		(0.46)	(0.71)	

TABLE II
COMPARISON TO PRIOR WORK [3]

	statistics in [3]		statistics of collapsed TLC (w/ ite)	
circuit	#AIG	#TLG	#AIG	#TLG
aes_core	20509	10057	19875	7753
wb_conmax	41070	21956	41163	19626
ethernet	57205	35243	43549	19110
des_perf	71327	42719	69500	27316
vga_lcd	88854	55402	90880	50734
geomean	(1.00)	(0.57)	(1.00)	(0.45)

the number of logic levels of the TL circuits, and the CPU time, respectively, for iterative collapsing with incrementing fanout bounds. The numbers in parentheses listed in Columns 6 and 9 (resp. Columns 7 and 10) are the ratios of gate counts (resp. level counts) of the collapsed TL circuits to those of original AIG circuits. Without incrementing the fanout bound iteratively, the collapse operation achieved an average of 50% reduction in gate count and 27% reduction in logic level; with the iterative collapse strategy, the reduction ratios were further enhanced to 54% in gate count and 29% in logic level.

To compare the synthesis quality to the method based on the merging operation proposed in [3], we also applied our iterative collapsing approach to the five largest benchmarks experimented in [3]. The results are shown in Table II. Columns 2 and 3, with data repeated from [3], report the numbers of AIG nodes in the benchmark circuits and the numbers of TLGs in the synthesized circuits. Columns 4 and 5 report the numbers of AIG nodes in the benchmark circuits (synthesized by `resyn2` script in `ABC`, as described in [3]) and the numbers of TLGs in the collapsed TL circuits. Apart from the slight mismatch between the numbers in Columns 2 and 4, an average of 43% reduction in gate count was achieved in [3] while an average of 55% reduction was achieved by our collapsing-based synthesis technique, confirming the generality and effectiveness of our formulation.

2) Collapsing Synthesized TL Circuits: To evaluate the room for collapsing on optimized TL circuits, we further conducted the following experiment. Benchmark circuits were first synthesized into TL circuits using the TL synthesis procedure (`&if` mapper in `ABC`) developed in [22]. The proposed collapsing-based synthesis method was then applied to the synthesized TL circuits.

Table III shows the results of collapse operation on TL circuits synthesized by [22]. Columns 4, 5, and 6 report the number of TLGs, the number of levels of the TL circuits synthesized by [22], and the CPU time, respectively; Columns 7, 8, and 9 report the number of TLGs, the number of logic levels of the TL circuits, and the CPU time, respectively, for direct collapsing; Columns 10, 11, and 12 report the number of TLGs, the number of logic levels of the TL circuits, and the CPU time, respectively, for iterative collapsing with incrementing fanout bounds. The numbers in parentheses listed in Columns 7 and 10 (resp. Columns 8 and 11) are the ratios of gate counts (resp. level counts) of the collapsed TL circuits to those of TL circuits optimized by [22].

On top of the synthesized circuits by [22], the collapse operation with the increment strategy obtained an average of 18% reduction in the number of TLGs efficiently (less than 6 s for the largest benchmark `b19`). The results show the effectiveness of the proposed collapse operation based on linear composition. The efficiency lies in fast analytic computation of collapsing conditions. However, the collapse operation achieved no reduction in logic level. This phenomenon could be attributed to the well-optimized `&if` mapper applied in [22]. Since the technology mapper has optimized the circuits in terms of delay, it is hard to further reduce logic levels.

In Fig. 4, we use the benchmark circuit `b19` as a representative to show the fanin size distribution of TLGs in TL circuits before and after iterative collapsing. Since the method in [22] can only synthesize TL circuits consisting of TLGs with no greater than eight fanins, there is no TLG with more than eight fanins before collapsing. Observe that the numbers of TLGs with less than 5 fanins decrease after collapsing, which is a general phenomenon among all large benchmarks, such as `b14` to `b22`. The proportion of the TLGs with more than eight fanins in large benchmarks accounts for over 13% of all gates. This notable proportion reflects the effectiveness of our collapse operation for TLG elimination.

B. Verification

1) Equivalence Checking Collapsed TL Circuits: To evaluate the performance of the proposed MUX/PB-based verification techniques discussed in Section VI-B,

TABLE III
STATISTICS OF COLLAPSING SYNTHESIZED TL CIRCUITS

benchmarks profile			statistics of TLC synthesized by [22]			statistics of collapsed TLC (w/o ite)			statistics of collapsed TLC (w/ ite)		
circuit	#pi	#po	#TLG	#level	time (s)	#TLG	#level	time (s)	#TLG	#level	time (s)
c6288	32	32	1424 (1.00)	29 (1.00)	5.04	1105 (0.78)	29 (1.00)	0.00	1102 (0.77)	29 (1.00)	0.10
c7552	207	108	950 (1.00)	10 (1.00)	3.45	716 (0.75)	10 (1.00)	0.00	713 (0.75)	10 (1.00)	0.06
s13207	31	121	1257 (1.00)	8 (1.00)	1.59	1046 (0.83)	8 (1.00)	0.00	1049 (0.83)	8 (1.00)	0.05
s15850	14	87	1630 (1.00)	10 (1.00)	2.31	1360 (0.83)	10 (1.00)	0.00	1356 (0.83)	10 (1.00)	0.06
s35932	35	320	5878 (1.00)	5 (1.00)	1.67	4299 (0.73)	5 (1.00)	0.01	4299 (0.73)	5 (1.00)	0.10
s38417	28	106	4857 (1.00)	8 (1.00)	4.43	4248 (0.87)	8 (1.00)	0.01	4220 (0.87)	8 (1.00)	0.22
s38584	12	278	4391 (1.00)	8 (1.00)	4.82	4000 (0.91)	8 (1.00)	0.01	3999 (0.91)	8 (1.00)	0.15
b14	32	54	2680 (1.00)	13 (1.00)	8.13	2253 (0.84)	13 (1.00)	0.01	2218 (0.83)	13 (1.00)	0.17
b15	36	70	4077 (1.00)	16 (1.00)	12.33	3616 (0.89)	16 (1.00)	0.01	3566 (0.87)	16 (1.00)	0.27
b17	37	97	13009 (1.00)	22 (1.00)	31.80	11279 (0.87)	22 (1.00)	0.04	11131 (0.86)	22 (1.00)	0.78
b18	37	23	36370 (1.00)	43 (1.00)	79.99	30732 (0.84)	43 (1.00)	0.13	30180 (0.83)	43 (1.00)	3.54
b19	24	27	72362 (1.00)	46 (1.00)	165.91	61320 (0.85)	46 (1.00)	0.29	60061 (0.83)	46 (1.00)	5.27
b20	32	22	5660 (1.00)	15 (1.00)	15.97	4679 (0.83)	15 (1.00)	0.02	4630 (0.82)	15 (1.00)	0.41
b21	32	22	5660 (1.00)	16 (1.00)	16.77	4730 (0.84)	16 (1.00)	0.02	4628 (0.82)	16 (1.00)	0.39
b22	32	22	8429 (1.00)	16 (1.00)	20.94	7000 (0.83)	16 (1.00)	0.02	6846 (0.81)	16 (1.00)	0.59
geomean			(1.00)	(1.00)		(0.83)	(1.00)		(0.82)	(1.00)	

TABLE IV
COMPARISON ON THE CNF-, MUX-, AND PB-BASED EQUIVALENCE VERIFICATION

Statistics for equivalence verification on TLC before and after collapsing												
	verification statistics of collapsed TLC (w/o ite)						verification statistics of collapsed TLC (w/ ite)					
	file size (MB)			verification time (s)			file size (MB)			verification time (s)		
circuit	CNF	MUX	PB	CNF	MUX	PB	CNF	MUX	PB	CNF	MUX	PB
c6288	0.5	0.0	0.3	TO	0.91	TO	0.4	0.0	0.3	TO	0.78	TO
c7552	0.2	0.0	0.2	0.56	0.18	1.89	0.2	0.0	0.2	0.63	0.17	2.24
s13207	0.4	0.1	0.5	0.00	0.08	1.54	0.4	0.1	0.5	0.00	0.07	1.38
s15850	0.5	0.1	0.5	0.00	0.25	1.70	0.5	0.1	0.5	0.00	0.21	1.84
s35932	1.2	0.2	1.5	7.49	0.55	15.47	1.2	0.2	1.5	7.13	0.55	19.49
s38417	1.5	0.2	1.5	10.57	0.47	23.15	1.4	0.2	1.5	11.07	0.47	24.63
s38584	1.5	0.2	1.5	0.01	0.73	22.09	1.5	0.2	1.5	0.01	0.64	28.22
b14	1.2	0.1	0.7	184.72	0.71	87.87	0.8	0.1	0.7	71.79	0.65	87.28
b15	1.6	0.1	1.2	9.86	1.10	14.34	1.4	0.1	1.2	9.37	1.47	16.87
b17	6.5	0.4	4.1	101.28	6.72	122.83	4.7	0.4	3.7	72.05	5.85	143.20
b18	18.8	1.1	11.3	TO	64.09	TO	13.8	1.1	10.3	TO	57.71	TO
b19	39.0	2.2	22.7	TO	260.82	TO	29.0	2.2	20.7	TO	272.99	TO
b20	2.4	0.1	1.5	57.41	1.78	1486.69	2.0	0.1	1.4	81.17	2.17	1929.11
b21	2.9	0.1	1.5	119.36	1.96	2632.56	2.0	0.1	1.4	58.05	2.04	1134.86
b22	4.1	0.2	2.2	99.10	2.80	2297.77	3.1	0.2	2.1	113.35	3.90	1505.68

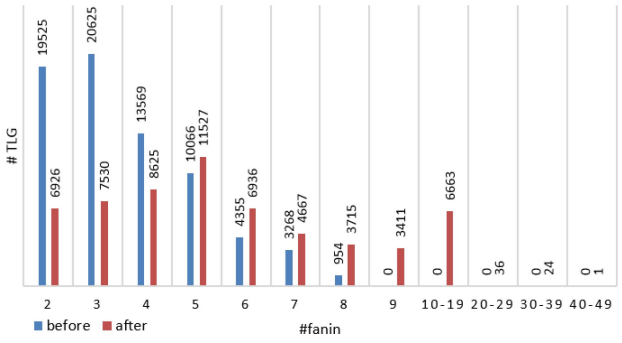


Fig. 4. Fanin size distribution of benchmark circuit b19.

we compared them against the state-of-the-art CNF-based verification method in [33], and used the equivalence checking between the pairs of TL circuits before and after collapsing as a case study. SAT solver Minisat (version 2.2.0), equivalence checking command cec in ABC, and PB solver Minisat+ (version 1.0) were adopted as the underlying constraint solving engines for CNF-, MUX-, and PB-based verification methods, respectively. As the translation time for TL circuits where most TLGs have small fanin sizes is insignificant in our experiment (e.g., less than a second to translate the largest benchmark circuit b19 into MUX trees), in the following we focus on the verification time, i.e., time spent on solving the constraints. The CPU time and memory space required to translate high-fanin TLGs will be studied later.

Table IV shows the results of equivalence verification between a TL circuit synthesized by the approach in [22] and its collapsed counterpart. The three verification techniques mentioned above, i.e., CNF-based, MUX-based, and PB-based, were evaluated in terms of CPU time and memory usage. Two different collapse strategies, direct collapsing, and iterative collapsing with incrementing fanout bounds, were compared. Columns 2, 3, and 4 (resp. Columns 8, 9, and 10) show the sizes in MB of the generated files for verifying the circuits not iteratively collapsed (resp. the circuits iteratively collapsed). In the cases of the CNF- and PB-based methods, the generated files are the input files to the SAT and PB solvers. For the MUX-based method, the generated files are pairs of AIG circuits derived from the MUX trees of TL circuits before and after collapsing. Columns 5, 6, and 7 (resp. Columns 11, 12, and 13) show the CPU time of equivalence checking (constrain solving) for the circuits not iteratively collapsed (resp. the circuits iteratively collapsed) by the three discussed techniques, respectively.

TABLE V
COMPARISON ON THE EFFECT OF PLAISTED-GREENBAUM ENCODING

	translation w/o PG encoding		translation w/ PG encoding	
circuit	#constr	solving time (s)	#constr	solving time (s)
b17_or	24086	1.32	16470	0.96
b18_or	70718	3.81	49061	2.58
b19_or	141582	7.69	98301	5.57
b17_and	23990	1.35	16426	0.95
b18_and	70254	3.82	48837	2.50
b19_and	140666	8.04	97864	5.25
geomean	(1.00)	(1.00)	(0.69)	(0.69)

We observe the following results from Table IV. First, in terms of the sizes of the generated files, the PB-based method in general produced smaller files compared to the CNF-based method due to the linear-time TL-to-PB translation. However, although the CNF-based method potentially suffers from the memory explosion issue when the fanin size of a TLG is large, for benchmark circuits we evaluated, the memory consumption was affordable. This is attributed to the fact that most TLGs in the evaluated TL circuits have small fanin sizes (usually less than 20). On the other hand, the MUX-based method achieved the most compact memory usage, due to the automatic logic sharing and simplification during the construction of MUX trees using the AIG package in ABC, and also the binary encoding scheme of AIGER format.

Second, while the CNF- and PB-based methods solved the same set of benchmark circuits, they failed to prove the equivalence for the largest benchmark circuits b18, b19, and the 32-b multiplier c6288. On the other hand, thanks to the efficient equivalence checking command cec in ABC, the MUX-based method was able to solve all the benchmark circuits in a few minutes. Notice that, it uniquely solved the largest benchmarks, demonstrating the great value in the proposed TL-to-MUX translation.

2) Effect of Plaisted-Greenbaum Encoding: As discussed in Section V-B1, the number of translated PB constraints can be reduced with Plaisted-Greenbaum encoding if some TLGs occur purely positively or negatively. However, in the previous equivalence checking experiment, there is no such TLG because the miter asserting equivalence consists of XORs and hence involves both polarities. Therefore, to study the effect of Plaisted-Greenbaum encoding, we have to consider other test cases. To demonstrate, we took circuits b14 to b22, and created two variants for each of them by disjoining (OR-ing) and conjoining (AND-ing) every PO. We synthesized the resulting circuit into a TL circuit, and translated it into PB constraints to test the satisfiability of its PO. For all circuits we tested, the PO of the disjunctive (resp. conjunctive) variant was satisfiable (resp. unsatisfiable). Table V shows the comparison between TLG-to-PB translations without and with Plaisted-Greenbaum encoding for the largest three benchmarks in our experiment. For both variants, Plaisted-Greenbaum encoding reduced the number of PB constraints and the solving time by about 31% on average.

3) Translation Scalability Versus Fanin Size: As mentioned above, most TLGs in the TL circuits verified in the previous equivalence checking experiment have small fanin sizes (usually less than 20). For the sake of a complete evaluation, we further studied the scalability of the compared TL-to-CNF, TL-to-MUX, and TL-to-PB translations as fanin size grows. Notice that, in practical neural network applications, it is common to have neurons with large fanin sizes, e.g., 64, 128, or even 256. Therefore, it is crucial to study the translation scalability with respect to fanin size to understand the feasibility of different approaches under practical neural network settings. To that end, we considered translating TLGs with fanin sizes in $\{16, 20, 24, 28, 32, 36, 40\}$. For each fanin size, we randomly generated 100 TLGs with weights and threshold values limited to $[-255, 255]$. The average sizes of generated files and the translation time for the three methods are reported in Fig. 5.

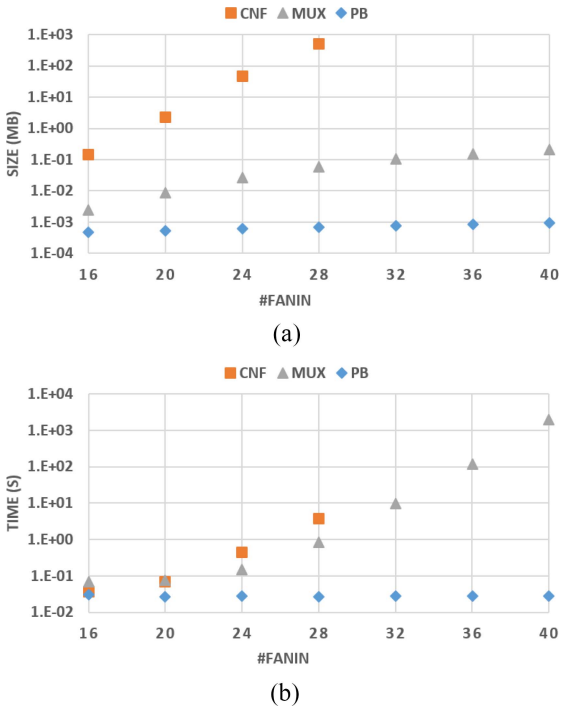


Fig. 5. Translation costs of high-fanin TLGs. (a) Average file size versus fanin size. (b) Average translation time versus fanin size.

As shown in Fig. 5(a), when the fanin size equals 28, the average size of the translated CNF formulas is close to the imposed memory limit (1GB). For TLGs with fanin sizes larger than 28, the sizes of generated CNF formulas exceeded the memory limit within roughly 10 s, and hence are not reported. On the other hand, although the sizes of the translated MUX trees did not exceed the memory limit, the translation time grows very quickly with respect to fanin size as shown in Fig. 5(b). For example, the average runtime to translate the randomly generated 40-input TLGs into MUX trees was about 2000 s.

Recall that in Table IV the MUX-based method has the most compact memory usage when verifying "TL circuits"; however, in Fig. 5(a) the average file size of the PB constraints translated from "a single TLG" is smaller than that of the MUX trees. We would like to clarify that these two observations do not contradict with each other. The advantage of the PB-based method for TLGs with large fanin sizes when applied to TL circuits is nullified by the facts: 1) Most TLGs

TABLE VI
PB-BASED EQUIVALENCE VERIFICATION OF BENCHMARKS FROM ACTIVATION-BNNS

#neuron input	accuracy (%)	file size (MB)	verification time (s)
32	92.95	0.14	25.65
48	94.37	0.30	103.64
64	94.79	0.53	288.16
128	95.98	2.10	21216.90

in the benchmarks evaluated in Table IV have small fanin sizes (e.g., roughly 90% of TLGs have fanin sizes less than 10 in `b19`); 2) the automatic logic sharing (AIG nodes in one MUX tree can be shared with other MUX trees) by the AIG package; and 3) the converted MUX trees are compactly represented with the binary encoding scheme used in AIGER format. On the other hand, there is no automatic logic sharing during the TL-to-PB translation (i.e., the constraints of different TLGs cannot be shared), and the constraints are represented in ASCII format, which consumes more memory. Therefore, the PB-based method did not achieve more compact memory usage for the evaluated benchmarks in Table IV.

Among the three translations, the proposed TL-to-PB translation achieved the best performance to translate high-fanin TLGs in terms of both space and time, thanks to its linear-time complexity with respect to fanin size. From the above scalability study, we conclude that only the proposed PB-based verification method is feasible for TL circuits consisting of high-fanin TLGs, which often arise from practical neural network settings.

4) Benchmarks From Neural Network Applications: To demonstrate the applicability of the proposed PB-based verification method to TL circuits derived from neural network applications, we trained activation-BNNs using the tool developed in [5] on the MNIST dataset [15]. Our network topology consists of one input layer, three fully connected hidden layers, and one output layer. The hidden layers were converted into TL circuits. Referring to the architecture parameters of TrueNorth chip [19], we scaled the floating-point weights into $[-255, 255]$ and $[-15, 15]$ to generate two variants of the TL circuits, and applied the proposed PB-based verification to check the equivalence between them.

Table VI shows the results. Four activation-BNNs with 32, 48, 64, and 128 neurons per hidden layer, respectively, were trained and their hidden layers were converted to TL circuits where the fanin size of each TLG equals the number of neurons per hidden layer. The accuracy of the neural networks and the time elapsed to check the equivalence between the two scaled variants are reported. For the four neural networks we evaluated, their scaled variants were functionally inequivalent. We remark that, although the equivalence of two neural networks can be tested with other more scalable methods, such as feed-forward simulation, our results show the possibility to apply formal methods developed for TL circuits to verify properties of practical neural networks.

To support the conclusion of the translation scalability study, we also tested the CNF- and MUX-based methods over the above neural network benchmarks. The MUX-based method was only able to verify the smallest network consisting of neurons with 32 inputs in 1075.25 s, generating AIG files of size 13.96 MB. On the other hand, the CNF-based method did not verify any network, always generating CNF files exceeding the imposed memory limit of 1 GB. These results confirm that among the evaluated approaches, only the proposed PB-based verification method is feasible for TL circuits arising from neural network applications, which stems from the linear-time TL-to-PB translation complexity.

C. Summary

We briefly summarize the implications of the proposed collapsing-based synthesis and the MUX/PB-based verification methods obtained from the above experimental results.

1) The collapsing-based synthesis is effective and efficient on both AIG circuits (Table I) and synthesized TL circuits (Table III). It outperforms the merging-based synthesis proposed in [3] (Table II), and further reduces the TLG counts of the synthesized TL circuits by 18% on average.
2) The MUX-based verification method achieves the best performance for the equivalence checking between the TL circuits before and after collapsing (Table IV). It uniquely verifies the largest benchmarks within a few minutes.
3) The TL-to-PB translation achieves the best translation scalability with respect to fanin size (Fig. 5), and the PB-based verification method is applicable to benchmarks derived from practical neural networks with up to 128 inputs per neuron (Table VI).

VIII. CONCLUSION

In this article, we studied the synthesis and verification of TL circuits through various kinds of constraint solving. For synthesis, the collapse operation was formulated, and the sufficient and necessary condition to collapse two TLGs using linear combination was derived. For verification, two novel techniques that translate a TL circuit into MUX trees and a set of PB constraints, respectively, were proposed. The resulting MUX-based verification method enjoys the well-developed approaches for Boolean logic circuits, and the PB-based one has a linear-time translation complexity with respect to the circuit size. Experimental results showed fast and effective TL circuit collapsing as well as efficient equivalence checking. Specifically, on top of the optimized TL circuits, an average of 18% gate count reduction was achieved by the collapse operation. For verification, the proposed MUX-based verification method verified the largest benchmarks uniquely in our experiments. Moreover, the PB-based verification method demonstrated its applicability to benchmarks arising from practical neural networks with up to 128 inputs per neuron. For future work, we would like to combine the collapse operation with other TL operations to form useful scripts for TL circuit optimization, to use the PB-based verification method for more neural network applications, and to compare our PB-based method against existing approaches for deep neural networks based on mixed integer linear programming.

ACKNOWLEDGMENT

The authors thank Chia-Chih Chi for preparing the benchmarks of activation-BNNs, Yi-Hsiang Lai for investigating the collapse operation, Alan Mishchenko for suggesting the MUX-based verification, and Augusto Neutzling for discussing prior work [22].

REFERENCES

[1] V. Beiu, J. M. Quintana, and M. J. Avedillo, "VLSI implementations of threshold logic—A comprehensive survey," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1217–1243, Sep. 2003.

[2] H. Chen, S.-C. Hung, and J.-H. R. Jiang, "Disjoint-support decomposition and extraction for interconnect-driven threshold logic synthesis," in *Proc. Design Autom. Conf.*, Las Vegas, NV, USA, 2019, pp. 73–78.

[3] Y.-C. Chen, R. Wang, and Y.-P. Chang, "Fast synthesis of threshold logic networks with optimization," in *Proc. Asia South Pac. Design Autom. Conf.*, Macau, China, 2016, pp. 486–491.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016. [Online]. Available: arXiv:1602.02830.

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

[7] T. Gowda, S. Leshner, S. Vrudhula, and G. Konjevod, "Synthesis of threshold logic circuits using tree matching," in *Proc. 18th Eur. Conf. Circuit Theory Design*, Seville, Spain, 2007, pp. 850–853.

[8] T. Gowda, S. Vrudhula, and G. Konjevod, "Combinational equivalence checking for threshold logic circuits," in *Proc. 17th ACM Great Lakes Symp. VLSI*, 2007, pp. 102–107.

[9] P. Gupta, R. Zhang, and N. K. Jha, "An automatic test pattern generation framework for combinational threshold logic networks," in *Proc. Int. Conf. Comput. Design VLSI Comput. Process.*, San Jose, CA, USA, 2004, pp. 540–543.

[10] G.-B. Huang, Q.-Y. Zhu, K.-Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?" *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 3, pp. 187–191, Mar. 2006.

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[12] H. Hulgaard, P. F. Williams, and H. R. Andersen, "Equivalence checking of combinational circuits using Boolean expression diagrams," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 7, pp. 903–917, Jul. 1999.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[14] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold logic circuits," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2011, pp. 396–403.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[16] 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 *Proc. Int. Conf. Comput.-Aided Design*, Austin, TX, USA, 2016, pp. 1–8.

[17] S.-Y. Lee, N.-Z. Lee, and J.-H. R. Jiang, "Canonicalization of threshold logic representation and its applications," in *Proc. Int. Conf. Comput.-Aided Design*, San Diego, CA, USA, 2018, pp. 1–8.

[18] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, Apr. 1987.

[19] P. A. Merolla *et al*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[20] A. Mishchenko, S. Chatterjee, J.-H. R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/ERL M92/41, 2005.

[21] S. Muroga, *Threshold Logic and its Applications*. New York, NY, USA: Wiley, 1971.

[22] A. Neutzling, J. M. Matos, A. I. Reis, R. P. Ribas, and A. Mishchenko, "Threshold logic synthesis based on cut pruning," in *Proc. Int. Conf. Comput.-Aided Design*, Austin, TX, USA, 2015, pp. 494–499.

[23] N. S. Nukala, N. Kulkarni, and S. Vrudhula, "Spintronic threshold logic array (STLA)—A compact, low leakage, non-volatile gate array architecture," *J. Parallel Distrib. Comput.*, vol. 74, no. 6, pp. 2452–2460, 2014.

[24] A. K. Palaniswamy and S. Tragoudas, "An efficient heuristic to identify threshold logic functions," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 8, no. 3, p. 19, 2012.

[25] A. K. Palaniswamy and S. Tragoudas, "Improved threshold logic synthesis using implicant-implicit algorithms," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 10, no. 3, p. 21, 2014.

[26] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *J. Symb. Comput.*, vol. 2, no. 3, pp. 293–304, 1986.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[28] E. M. Sentovich *et al*. "SIS: A system for sequential circuit synthesis," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/ERL M92/41, 1992.

[29] J. L. Subirats, J. M. Jerez, and L. Franco, "A new decomposition algorithm for threshold synthesis and generalization of Boolean functions," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 10, pp. 3188–3196, Nov. 2008.

[30] C.-K. Tsai, C.-Y. Wang, C.-Y. Huang, and Y.-C. Chen, "Sensitization criterion for threshold logic circuits and its application," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2013, pp. 226–233.

[31] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning*. Heidelberg, Germany: Springer, 1983, pp. 466–483.

[32] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Synthesis and optimization of threshold logic networks with application to nanotechnologies," in *Proc. Design Autom. Test Eur. Conf.*, Paris, France, 2004, pp. 904–909.

[33] Y. Zheng, M. S. Hsiao, and C. Huang, "SAT-based equivalence checking of threshold logic designs for nanotechnologies," in *Proc. 18th ACM Great Lakes Symp. VLSI*, 2008, pp. 225–230.

[34] *Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification*. [Online]. Available: http://www.eecs.berkeley.edu/∼alanmi/abc/

Nian-Ze Lee (Student Member, IEEE) received the bachelor's degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2014, where he is currently pursuing the Ph.D. degree with the Graduate Institute of Electronics Engineering, supervised by Prof. J.-H. R. Jiang.

He also had internship experience with IBM T. J. Watson Research Center, Ossining, NY, USA, the National Institute of Informatics, Tokyo, Japan, and Ludwig Maximilian University of Munich, Munich, Germany. His research mainly focuses on the optimization and validation of computational systems with formal methods.

Jie-Hong R. Jiang (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1996 and 1998, respectively, and the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 2004.

During his compulsory military service, from 1998 to 2000, he was a Second Lieutenant with Air Force. He is a Professor with the Department of Electrical Engineering, Graduate Institute of Electronics Engineering, and Genome and Systems Biology Degree Program, National Taiwan University, Taipei, Taiwan. He leads the Applied Logic and Computation Laboratory, and has worked extensively on logic synthesis, formal verification, electronic design automation, and computation models of biological and physical systems.

Dr. Jiang is a member of Phi Tau Phi and the Association for Computing Machinery.