Effective Logic Synthesis for Threshold Logic Circuit Design

Augusto Neutzling, Jody Maick Matos,
Alan Mishchenko, Andre Reis, and Renato P. Ribas

***Abstract*—This paper presents a novel and effective logic synthesis flow able to identify threshold logic functions during the technology mapping process. It provides more efficient logic covering, exploring also redundant cuts. Moreover, the proposed design flow takes into account different circuit area estimations, such as the sum of input weights and threshold values, the gate fanin and the number of threshold logic gates. As a result, the mapped circuits present a reduction up to 47% and 67% in area and logic depth, respectively, in comparison to the most recent related approaches.**

***Index Terms*—threshold logic, technology mapping, emerging nanotechnologies, digital circuit, logic synthesis.**

I. Introduction

Threshold logic is a powerful alternative paradigm for implementing Boolean functions in digital circuit design. A threshold function can be roughly defined as a Boolean function in which the output is evaluated in terms of input weights and a given function threshold value. Although this subject has been studied since the 1960's, the lack of effective hardware construction of threshold functions led to a loss of interest in developing specific circuit design flow and methodologies, comprising specialized algorithms and CAD tools. However, more recently, some emerging technologies, such as memristor [1][2], quantum cellular automata (QCA) [3], magnetic tunnel junction (MTJ) [4][5] and resonant tunneling device (RTD) [6], have demonstrated to be more appropriate for threshold logic than for the most conventional switch-based CMOS/FinFET circuitry, which leads to AND/OR based synthesis.

As a consequence, algorithms addressing threshold logic synthesis have been presented in the literature [7]–[13]. The major drawback of these methods is the fact that they do not consider threshold logic while generating an initial covering in terms of 6-input single-output nodes. In general, once the circuit has been covered, these previous approaches generate locally a threshold network for each single-output node, aiming to cover the circuit using only threshold logic gates (TLGs). Moreover, the area estimation relies only on the number of TLGs in the final circuit. However, such an estimation may be inaccurate, since some TLG-based designs fit better when considering the total sum of input weights and threshold values, or even the total number of gate inputs (fanin) [1][4][6].

In this paper, we propose a novel threshold logic circuit design flow comprising several orthogonal improvements to TLG-oriented technology mapping. (1) By exploiting the thresholdness information from quite effective threshold logic identification, the proposed flow is able to combine area and circuit logic depth optimization through a simple and fast procedure with good quality-of-results (QoR). (2) By exploiting redundant cuts, the number of enumerated threshold cuts is increased enabling the covering to reduce even more the circuit area. (3) The proposed approach is able to target different threshold-based area estimations, such as the sum of input weights and threshold values, the gate fanin and the number of TLGs, allowing the exploitation of circuit design particularities related to different threshold-oriented emerging technologies.

The proposed flow has been implemented in the ABC logic synthesis tool [14]. The experimental results demonstrate the gains of the proposed threshold-oriented synthesis against the standard AND/OR-based flow provided in current commercial EDA tools. We also compare the results with the most relevant methods for threshold synthesis, in terms of circuit area and logic depth. In order to demonstrate the scalability of this method, we have carried out the synthesis of benchmark circuits comprising millions of gates. We also demonstrate that the proposed approach is able to treat TLGs with large fanin, whereas other methods are limited to six inputs.

The rest of the paper is organized as follows. In Section II, some fundamentals are briefly reviewed for a better understanding of the proposed approach. Section III discusses some related works. The proposed threshold synthesis flow is described in Section IV, presenting the improvement by enumerating redundant cuts and the proposed strategy for a technology-oriented area mapping. Section V provides the experimental results, whereas conclusions are given in Section VI.

II. Preliminaries

In this section, some fundamentals on Boolean networks, structural cuts and threshold logic are presented. Different area estimations taken into account in this work are discussed. We

A. Neutzling, J. M. Matos, A. Reis and R. P. Ribas are with the Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), 9500 Porto Alegre, Brazil (e-mail: {ansilva, jody.matos, andre.reis, rpribas}@inf.ufrgs.br).

A. Mishchenko is with the Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, CA 94720 (e-mail: alanmi@eecs.berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.XXXXXXX

also review the LUT-based technology mapping for FPGAs, which is the basis of the proposed mapper.

A. General Terms and Definitions

A Boolean network is a directed acyclic graph (DAG) where nodes correspond to logic gates and directed edges represent wires connecting the gates. It is assumed that each node has a unique ID (integer number). A *fanin* (*fanout*) cone of node n is a subset of all nodes of the network reachable through the *fanin* (*fanout*) edges from the given node. A node n has zero or larger *fanin* (nodes driving n) and zero or larger *fanout* (nodes driven by n). The primary inputs (PI) are nodes without *fanin*, whereas the primary outputs (PO) are a subset of nodes from the network connecting it to the environment.

AND-Inverter graph (AIG) is a particular DAG where each node has either zero incoming edges (in the case of PI) or two incoming edges (in the case of AND node). Each edge can be complemented or not. Some nodes are marked as PO.

Moreover, in terms of *structural cuts*, a *cut* c of a node n is a set of nodes in the network, called *leaves* of the cut, such that every path between a PI and n contains a node in c. A cut of n is irredundant if no subset on it is a cut. A K-feasible cut contains K or fewer nodes. Node n is called the root of cut c. The cut size is the number of its leaves. A trivial cut is the node itself. A local function of an AIG node n, denoted by $f_n(x)$, is a Boolean function of the logic cone rooted in n and expressed in terms of the leaves x of a cut of n. Cut enumeration is a technique used by a cut-based technology mapper to perform cut computation using dynamic programming, starting from PI and ending at PO [15][16].

Furthermore, in terms of classes of functions, a set of all functions with up to n variables can be grouped taking into account the negation (N), and/or the permutation (P) of variables, and/or the negation of the function value, so creating *NPN representatives* [17]. For instance, NP-class corresponds to the set of distinct functions obtained by negating and/or permuting the input variables. Similarly, NPN-class corresponds to the set of distinct functions obtained by negating and/or permuting the input variables and/or negating the function value.

B. Threshold Logic Terms and Area Estimation

A *threshold logic function* (TLF) is a Boolean function satisfying the following condition [18]:

$$f = \begin{cases} 1, & \textit{if} \sum_{i=1}^{n} x_i w_i \geq T \\ 0, & \textit{otherwise} \end{cases} \tag{1}$$

where x_i represents each Boolean input value $\{0, 1\}$, w_i is the weight of each input, and T is the function threshold value. Therefore, each input has a specific weight and the function has a threshold value. If the sum of active input weights (*i.e.*, inputs are equal to 1) is greater or equal than the threshold value, then the function evaluates to 1. Otherwise, the function evaluates to 0.

A TLF is completely represented by a compact vector [w_1,w_2,…,w_n;T], where w_1,w_2,…,w_n are the input weights and T is the function threshold value. For instance, the corresponding TLG of the given functions $f = x_1x_2x_3$ and $g = x_1 + x_2 + x_3$ are $[1, 1, 1; 3]$ and $[1, 1, 1; 1]$, respectively. A TLF has also been called ‘linearly separable’ function.

Although some complex functions are TLFs, there exist some simple functions that are not TLF. For instance, the function $h = x_1x_2 + x_3x_4$ cannot be represented in terms of input weights and threshold value. The *threshold logic identification* process verifies if a given Boolean function is TLF (or not) and compute the input weights and the corresponding threshold value. In this work, we have adopted the identification process presented in [19], instead of the integer-linear-programming (ILP) based algorithms applied in previous works [7]–[10], [20]. This is mainly due to the fast runtime and good quality of results (QoR) obtained. In such an approach, a complete system of inequalities is also built using a similar strategy to ILP inequalities generation algorithms. However, unlike ILP-based approaches, the inequalities system is not solved. Instead, the algorithm speeds up the process by selecting some of the inequalities as constraints to the associated variables and computing the variable (input) weights in a bottom-up strategy. After this assignment, the consistency of the entire system is verified in order to check whether the weights have been correctly computed.

In the context of structural cuts, we call a *threshold cut* the cut that the corresponding Boolean function is TLF.

Threshold logic gate (TLG) is a single primitive or a non-decomposable circuit that physically embodies the behavior expressed in Equation (1). TLGs can represent the implementation of complex functions. For instance, the TLG defined by $[4, 3, 3, 1, 1; 7]$ corresponds to $f = x_1x_2 + x_1x_3 + x_2x_3x_4 + x_2x_3x_5$. Using larger threshold functions has the potential benefit of reducing the total number of gates needed to implement digital circuits. Several topologies of TLGs have been proposed for CMOS and emerging nanotechnologies. A survey with more than 50 TLG circuitries are presented by Beiu *et. al*, in [21]. Moreover, TLG designs based on memristors [1], spintronics devices [4] and RTDs [7] have also been proposed.

Threshold logic network (TLN), on the other hand, is a netlist of TLFs and interconnections. A TLN can be implemented using TLGs, since each TLF can be directly implemented through a single TLG. The area of TLN corresponds to the sum of the TLG areas. Notice that the TLG area depends directly on the technology adopted to build such a gate. However, since no threshold-oriented technology currently available is mature enough to fabricate reliable TLGs in large scale, synthesis tools usually consider that each threshold gate presents the same area. Therefore, the total circuit area is usually estimated through the overall number of TLGs instantiated in the mapped circuit.

Although it is hard to define a general TLG area estimation, some of them are more suitable for specific technologies. For instance, when designing a TLG by using RTDs, each input weight and function threshold values determine the diode physical area. Therefore, the gate area is directly related to the sum of the input weights and the threshold value, as follows

[7]:

$$A_{\text{TLG}} = T + A_u(\sum_{i=1}^{k} w_i) \tag{2}$$

where k is the number of TLG inputs (fanin), w_i is the weight of input i, A_u is the unit area of an RTD with w=1, and T is the threshold of the gate.

In other technologies, such as memristors [1][2] and spintronic-based devices [4][5], the input weight is set by applying a voltage over the device during a moment, so not impacting the device physical dimensions. In these cases, each input is associated to a single device (with the same area) and, as a consequence, the most appropriate gate area estimation metric is the number of gate inputs.

C. LUT-based Technology Mapping

Common area and delay design cost criteria for threshold logic synthesis are, respectively, the TLG count and the circuit logic depth. Therefore, it is straightforward to relate threshold logic synthesis with LUT-based technology mapping.

The technology mapping process transforms a technology-independent logic network, named *subject graph*, into a network of logic nodes. In SRAM-based FPGA, each logic node is represented by a K-input LUT implementing any Boolean function up to K variables [15]. The subject graph is often represented as an AIG.

The performance of FPGA-based circuit is determined by two factors, the delay in K-LUTs and the delay due to the interconnection wires. Each K-LUT has a constant delay independent of the function configured, which represents the access time of a K-LUT. The interconnection delay is dominated by the physical configuration of the FPGA architecture, which is not available during the logic synthesis. Thus, LUT-based technology mappers assume that each edge in the mapping solution has a constant delay. Due to these reasons, the circuit delay is commonly represented by the circuit logic depth and the circuit area is represented by the number of K-LUTs from the mapped solution.

Technology mappers for FPGA produce near-optimum logic depth while minimizing the number of LUTs in the resulting network [15][22][23]. A typical procedure comprises the following steps:

1) Near-optimum delay mapping: computes arrival time at each node by computing the depth of priority cuts and choosing the best one;
2) Area recovery: reduces area using several heuristics, for instance, area flow and exact local area [14];
3) Choose the resulting cover.

Notice that the logic depth optimality is only achievable by this procedure because the delay-oriented mapping problem has two important properties: (*i*) its optimal solution can be obtained by using the optimal solution from its subproblems; and (*ii*) its subproblems overlap, *i.e.*, solutions of same subproblems are repeatedly needed. With this, through dynamic programming, choosing the best cut of a node is incrementally solving the mapping problem from inputs to outputs.

Moreover, circuit logic depth optimality requires the computation of all cuts in delay mapping, which is not scalable, *i.e.*, the number of K-cuts in a given network with n nodes is $O(n^K)$. In order to avoid exhaustive computation, the concept of priority cuts allows for the computation of a small number C of K-cuts at each node (typically $4 \leq C \leq 8$). The priority cut computation does not guarantee depth optimality. However, the circuit logic depth is optimal in 95% of the cases [24]. This one-pass, depth-oriented mapping increases substantially the number of LUTs, which is brought down on the area recovery step [15][22].

III. Related Works

The goal of threshold logic synthesis is to generate a TLN where each TLF can be directly implemented through a single TLG. The design flow adopted by previous works starts by performing a LUT-based technology mapping [7]–[11]. This first mapping task results in a netlist of Boolean functions with restricted fanin. Afterwards, these methods identify which Boolean functions are TLF and then include them into the final solution. For each non-TLF, they generate a sub-network. The main differences among these approaches are: (i) the procedure of the threshold logic identification and (ii) the generation of the sub-TLN from a non-threshold function.

Zhang *et al.*, in [7], and Subirats *et al.*, in [8], use ILP to perform the TLF identification. For non-TLFs, the Zhang's method decomposes the function into AND/OR sub-fuctions, which are always TLFs, and select nodes through a heuristic way in order to combine them, checking whether the resulting function is TLF. The Subirats' approach, on the other hand, is based on the function truth table description. It selects recursively a variable and performs the Shannon decomposition up to find TLF sub-functions. The Subirats' method improves the Zhang's results in terms of the number of gates and circuit logic depth. However, the Subirats' approach produces only two-level TLNs without fanin restrictions, being more suitable to neural network design.

In [9], Gowda *et al.* propose a heuristic approach to identify TLF. They adopt both binary decision diagrams (BDD) and a factorized tree structure (called max literal factor tree - MLFT) in order to generate a TLN. The method breaks recursively the initial expression tree into sub-expressions, identifying subtrees that represent TLF and assigning input weights. The method proposed by Palaniswamy *et al.*, in [10], improves the Gowda's approach [9]. It looks for circuit outputs that can be implemented as a single TLF. Both the Gowda's and the Palaniswamy's methods suffer from execution time as the main bottleneck, and the solutions depend strongly on the initial structure (BDD or MLFT), in particular, on the ordering of tree nodes.

The method proposed by Neutzling *et al.*, in [11], is based on a TLG association process through the principle called functional composition, which is based on dynamic programming [26]. The algorithm associates simpler sub-solutions, with known design costs, *e.g.* the number of gates, in order to produce the final solution with minimum cost. In order to identify whether the Boolean function created from such an association is TLF, the method adopts the heuristic method proposed in [27]. The Neutzling's approach presents improved results in terms of TLG count when compared to previous approaches. However, the design optimization in respect to the circuit logic depth is not so significant. Moreover, the execution time is also a limitation and the approach does not scales for TLFs with more than six variables (inputs).

The threshold logic synthesis methods mentioned above generate a TLF network from a generic Boolean function netlist. Another set of approaches focuses in the optimization starting from a TLN. Methods for TLG-based circuit rewiring are presented by Kuo *et al.*, in [28], and by Lin *et al.*, in [12]. Kuo's approach focuses only on circuit restructuring for satisfying a new fanin constraint and does not take into account the area and logic depth minimization issue. Lin's approach, on the other hand, represents a heuristic for rewiring the circuit by minimizing the summation of input weights and threshold value. In [13], Chen *et al.* propose an analytical approach based on collapsing two threshold gates in order to minimize the total number of TLGs. Annampedu's method, in [29], receives as input a given (already identified) single-output threshold function. Therefore, this method is not able to treat a general logic circuit with multiple outputs corresponding to Boolean functions not necessarily identified as threshold ones. The main goal of Annampedu's method is to restrict the fanin of a given threshold function implementation. Finally, Kulkarni *et al.*, in [30], propose TLG-based approaches to reduce the circuit area and power consumption without loss of performance. However, the technology mapping is performed by a commercial tool using conventional (*i.e.*, non-threshold based) standard cell library. This method replaces some standard flip-flops by threshold logic sequential cells [30]. As a consequence, a hybrid netlist comprising both TLGs and conventional logic gates is provided.

Notice that all of mentioned threshold logic synthesis approaches focus basically on synthesizing single-output non-TLF. The first step of previous methods relies on a complete synthesis process which disregards threshold logic domain. Therefore, they do not explore the entire circuit where they would find, for instance, TLFs between different functions in the netlist.

IV. Proposed Threshold Logic Synthesis

Given a digital circuit functionality, usually described in RTL format, the threshold logic synthesis (TLS) relies on finding an optimized threshold logic network. The most TLS tools initially cover the circuit by single-output nodes, disregarding their thresholdness, as illustrated in Fig. 1(a) [7]–[11]. Then, these approaches perform the resynthesis of any non-TLF found to individual, unshared sub-TLN in order to provide the covering of the final threshold logic network.

In [25], we propose a logic synthesis flow that identifies TLFs before the circuit covering task. It is based on a three-stage procedure, as depicted in Fig. 1(b): (*i*) a complete cut enumeration, storing Boolean functions of cuts in the design; (*ii*) the identification of TLFs related to this set of computed cuts; and (*iii*) the technology mapping considering thresholdness of pre-computed functions. By doing so, this approach is able to discard non-TLF cuts and provides the corresponding threshold network from the first covering action. Such a strategy allows the exploitation of multi-objective technology mapping algorithms, as described in Section II-C. On the other hand, notice that this approach relies on computing cuts twice in the flow, in the stages (*i*) and (*iii*), mentioned above.

A. Unified Threshold Logic Synthesis Flow

The new proposed TLS flow, illustrated in Fig. 1(c), seeks a single-stage approach, so avoiding to compute cuts twice in the flow and performing the threshold logic identification during

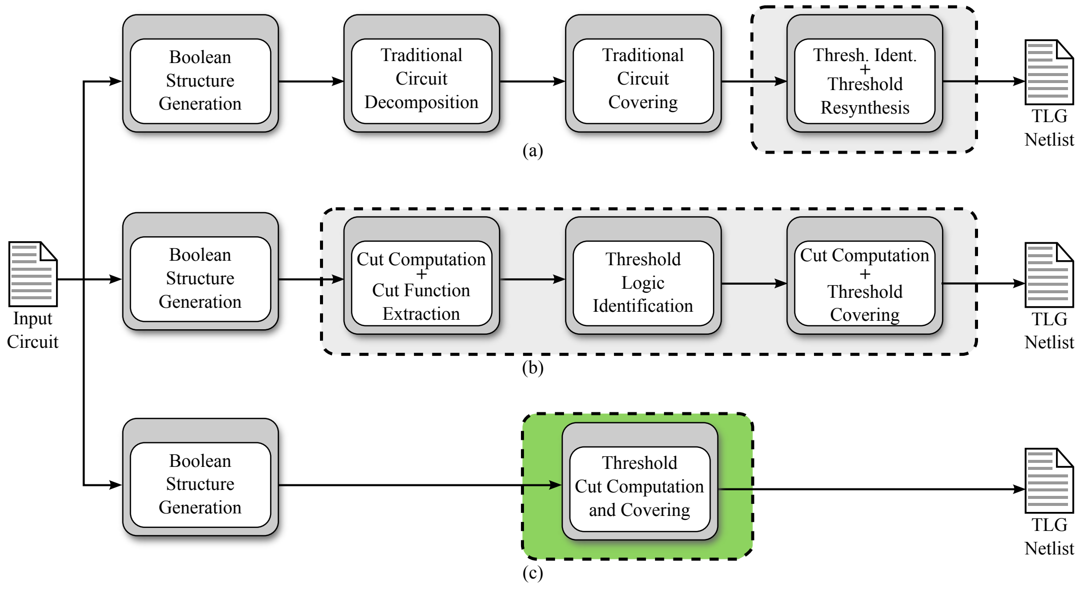
Boolean Structure Generation
Traditional Circuit Decomposition
Traditional Circuit Covering
Thresh. Ident. + Threshold Resynthesis
TLG Netlist
(a)
Input Circuit
Boolean Structure Generation
Cut Computation + Cut Function Extraction
Threshold Logic Identification
Cut Computation + Threshold Covering
TLG Netlist
(b)
Boolean Structure Generation
Threshold Cut Computation and Covering
TLG Netlist
(c)

Figure 1. Different threshold logic synthesis flows: (a) previous works from other authors, (b) proposed in [25]; (c) proposed in this work.

the technology mapping task. A pseudocode is presented in Algorithm 1 and is explained in the following.

Initially, the AIG is traversed in topological order (from primary inputs to primary outputs). For each visited node, an empty cut set is created (line 2 in the Algorithm 1). Then, the cuts of a node are computed as a Cartesian product of the cuts from its children (lines 3-5). Notice that only cuts that respect the fanin constraint are considered in the procedure (lines 6-7).

In the core of the cut computation task, we propose to extract the Boolean function of each cut and to compute an NPN representative of the target function (lines 8-16) [17]. By making so, we guarantee that the threshold identification is performed only once per NPN representative through the use of hash table. This procedure aims to ensure that only threshold cuts are matchable in the covering task.

The next step is to add the new cut into the node cuts set. As we have adopted the priority cut strategy (see Section II-C), the cut set size is limited by a user-defined parameter C, being typically eight, and the set is kept ordered at each iteration. Common ordering criteria and tie-breaks are area, delay and fanin. If the cut set is larger than C, then the worst cut is discarded.

Notice that the thresholdness information is already available when adding the new cut into the cut set. This fundamental change, along with the important role of the best cut of a node, can be used to guarantee that only threshold cuts are taken into account during the mapping process. Moreover, at the end of the process, the trivial cut (*i.e.*, the node itself) is added to the cut set. Once AIG have been adopted as subject graph, the entire design is already decomposed in AND nodes, which are TLFs. By adding the trivial cut, we ensure that the proposed method can always find a TLF-only covering.

Once the cut computation step is finished, a covering step can traverse the graph in reverse topological order and recursively selects the best cut fanins so that it covers the entire graph. Since all the best cuts are threshold, the resulting mapping only comprises cuts that can be expressed through TLFs.

The complexity of the technology mapping task is $O(KnC^2)$, dominated by the cut computation that is linear in the size K of cuts and in the number of circuit nodes n, and quadratic in the number of cuts C stored at each node. The matching of cut functions to TLFs is performed in constant time for each cut by using a hash lookup. The complexity of the identification step is $O(Klog(K)mm')$, being m and m' the number of prime implicants related to the on-set and off-set of the TLF candidate, respectively. As TLF is a unate function, the number of prime implicants m (or m') is at most $\frac{K!}{\lfloor K/2 \rfloor ! \cdot \lceil K/2 \rceil !}$. For additional information about number of prime implicants refer to [31]. Notice that these time complexities are bearable for small K, up to 15 inputs. Additionally, the identification step is performed only once per NPN representative of each function.

Algorithm 1: Pseudocode of proposed approach.

```
Input: $AIG$, $K$, $C$
Output: $AIG$ with covering information
/* traverse and nodes in topo order */
1  foreach $node \in AIG.andNodes$ do
2      $cutSet$ = empty cut set;
3      foreach $cut0 \in node.fanin0.cutSet$ do
4          foreach $cut1 \in node.fanin1.cutSet$ do
5              $cut$ = merge($cut0$,$cut1$);
6              if $cut.nLeaves > K$ then
7                  continue;
8              $cutFunc$ = extracted function from $cut$;
9              $cutNpn$ = NPN representative of $cutFunc$;
10             if $cutNpn$ is hashed then
11                 $cut.isThreshold$ = get from hashtable;
12                 $cut.thresholdArea$ = get from hashtable;
13             else
14                 $cut.isThreshold$ = run identification;
15                 $cut.threshArea$ = get from identification;
16                 add this information into the hashtable;
17             add $cut$ into $cutSet$ and sort it;
18             if $cutSet.nCuts > C$ then
19                 discard the worst cut $\in cutSet$;
20     add the trivial cut into $cutSet$ and sort it;
21     $node.bestCut$ = the best threshold cut in $cutSet$;
```

B. Improvements by Enumerating Redundant Cuts

In the context of logic synthesis, the state-of-the-art algorithms for cut computation rely on enumerating irredundant cuts only [15]. By pruning redundant cuts, the enumeration methods have a better performance, both in terms of runtime and memory usage, without loss of QoR for general cases. For instance, near-optimal circuit logic depth is still achievable. This procedure is the basis of methods for cut computation, such as the priority cuts approach presented in [24].

In this paper, we demonstrate that cleverly relaxing the cut redundancy checking may decrease the circuit area without significant impact neither on the circuit performance nor on the method runtime. The following example, depicted in Fig. 2, illustrates such an improvement.

Consider the case in which the AIG depicted in Fig. 2(a) is taken as input in a threshold logic technology mapping. If only irredundant cuts are enumerated, then the most likely covering instantiates three TLGs in the mapped circuit, as illustrated in Fig. 2(b). The cuts considered in this covering are $x = \{A, B\}$, $z = \{A, B, C\}$ and $w_i = \{x, z\}$. On the other hand, if redundant cuts are allowed to be additionally enumerated, a resulting covering comprising only two TLGs is achieved, as depicted in Fig. 2(c). In this case, the cuts used in the covering are $x = \{A, B\}$ and $w_r = \{x, A, B, C\}$. Notice that the cut w_r would never be enumerated in the prior mapping, since it is redundant with respect to the cut $z = \{A, B, C\}$ because w_r is a superset of z. However, computing the redundant cut w_r allowed to decrease the number of TLGs during the mapping procedure from three to two, and without impacting the circuit logic depth.

The idea of looking for redundant threshold cuts has been

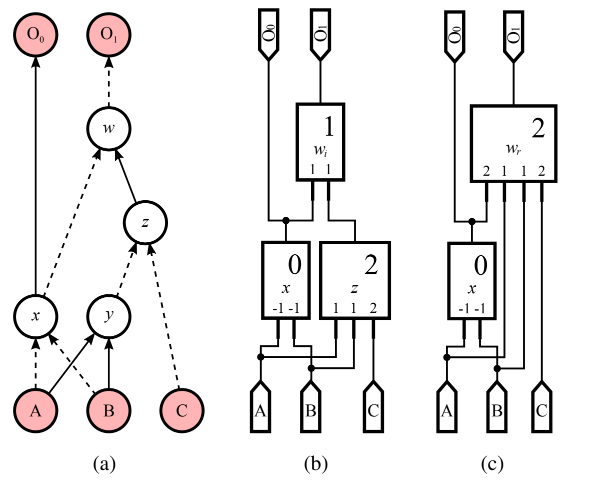

Figure 2. Example of area reduction by enumerating redundant cuts: (a) AIG; (b) mapped circuit by enumerating only irredundant cuts; (c) mapped circuit with reduced area by enumerating redundant cuts.

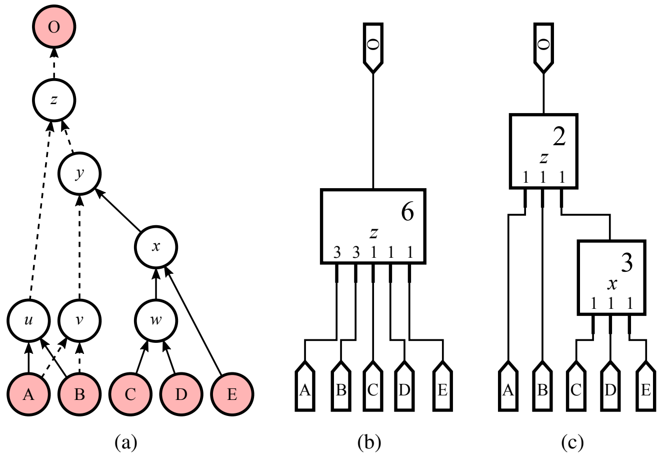

Figure 3. Example of TLNs obtained by taking into account different area estimations: (a) a given AIG; (b) mapped circuit optimizing the TLG count; (c) mapped circuit optimizing the summation of input weights and threshold values.

primarily mentioned by Kulkarni and Vrudhula, in [32]. In that work, the authors claim that, despite some threshold cuts are not enumerated for being redundant, including redundant cuts in the enumeration procedure tends to result in more computational requirements. In order to overcome such a bottleneck, they propose to change from node cut to line cut computation strategy and to compute only unidirectional cuts. By making so, the relationship between strong line cuts in a DAG and independent sets is explored [33].

In order to avoid such a changing on the cut computation paradigm but still trying to keep the procedure computationally feasible, we propose to relax the cut redundancy checking while enumerating priority cuts. As a consequence, we allow for redundant threshold cuts to be added in the priority cut set according to the sorting function. Such a strategy is possible since the threshold identification is performed during the cut enumeration.

C. *Effective Mapping for Different TLG Area Estimation*

To the best of our knowledge, related works on threshold logic synthesis are restricted in optimizing the overall number of TLGs [7]–[11][13]. Although some of these works take into account different area estimations for TLG-based circuit, they still focus on the optimization of the total number of TLGs during the synthesis. In [7], the authors just report those numbers with respect to the final netlists. In [12], Lin *et al.* adopt the Zhang's approach in order to obtain a threshold logic network, and the authors propose a method of rewiring to locally improve the summation of input weights and threshold value.

We propose a new effective technology mapping that is able to optimize different area estimations for threshold logic based circuit. The area estimations considered herein are the overall number of TLGs, the sum of input weights and threshold values, and the gate fanin. These parameters are more suitable to up-to-date TLG physical implementations, as discussed in Section II-B.

In order to optimize different circuit area estimations, the TLG area is computed along with the threshold identification procedure. As discussed in Section IV-A, the threshold identification is performed only once per NPN representative during the cut enumeration process. If the identified function is threshold, the corresponding TLG area is computed and stored in a hash table. Afterwards, this information is used during the covering task.

In the following, it is described a case where the optimization of the suitable design cost function impacts on different mapping results. Given the functionality represented by the AIG depicted in Fig. 3(a), the whole circuit can be implemented by a single, 5-input TLG. Such a solution is optimal in terms of TLG count. However, it does not represent the best solution in terms of final circuit area whether the related area estimation is defined as the overall summation of input weights and threshold value. For instance, considering such an area estimation, the solution comprising just a single TLG, depicted in Fig. 3(b), presents the total circuit area equals to 15. In contrast, the best solution in terms of input weights and threshold value is obtained by instantiating two TLGs, as illustrated in Fig. 3(c), and the corresponding total area is equal to 11. Both solutions are found by the proposed method by just configuring the most appropriate area cost function.

V. Experimental Results

In order to validate the proposed threshold logic synthesis approach, different experiments were carried out taking into account three sets of benchmark circuits. The first set aims to support the claims stated in Section IV, so comparing the results to the previous approach presented in [25]. In the second set, the proposed work is compared to related threshold logic synthesis approaches from other authors [7]–[13]. In the third set of experiments, we present a comprehensive collection of results when customizing the developed mapper to achieve different mapping goals (*i.e.*, circuit area versus logic depth) and targeting different TLG-based circuit area

Table I
SUMMARY OF RESULTS FROM THE PROPOSED THRESHOLD LOGIC SYNTHESIS FLOW IN COMPARISON TO [25].

	Approach presented in [25]						Proposed work		
circuit	*#TLGs*	*logic depth*	*step 1 (sec)*	*step 2 (sec)*	*step 3 (sec)*	*total runtime (sec)*	*#TLGs (ratio)*	*logic depth (ratio)*	*total runtime (ratio)*
oc_aquarius	9271	29	16.77	0.81	17.10	34.68	(0.99)	(1.00)	(0.63)
oc_cfft_1024x12	4002	8	6.28	0.86	6.90	14.04	(0.99)	(1.00)	(0.76)
oc_cordic_p2r	4007	8	5.90	0.95	6.82	13.67	(0.98)	(1.00)	(0.73)
oc_cordic_r2p	4943	8	6.58	1.00	7.12	14.70	(0.98)	(1.00)	(0.74)
oc_des_perf	9245	7	26.79	0.19	27.97	54.95	(0.99)	(1.00)	(0.79)
oc_ethernet	3741	7	3.96	0.16	4.24	8.36	(0.99)	(1.00)	(0.61)
oc_fpu	8501	281	19.44	1.30	18.31	39.05	(0.96)	(1.00)	(0.66)
oc_mem_ctrl	6626	8	5.97	0.08	6.59	12.64	(0.99)	(1.00)	(0.59)
oc_video_dct	14748	13	34.22	1.24	32.36	67.82	(0.99)	(1.00)	(0.62)
oc_video_jpeg	19218	12	31.58	1.19	31.58	64.35	(0.98)	(1.00)	(0.66)
radar20	30745	14	73.96	2.12	65.38	141.46	(0.99)	(1.00)	(0.47)
uoft_raytracer	61879	23	210.75	3.17	182.54	396.46	(0.98)	(1.00)	(0.49)
geomean							(0.98)	(1.00)	(0.64)

estimations. All results have been checked for combinational equivalency by using the ABC "*cec*" command [14]. For the sake of reproducibility, all the experiments can be repeated using the complete dataset available in [34], which includes the optimized benchmarks and threshold synthesis scripts.

The proposed approach has been implemented in the ABC tool [14], using C programming language and compiled using *gcc 4.7.2*. The experiments were performed on a computer with Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz, 8GB RAM.

A. Evaluation of Proposed Improvements

First of all, we compare the new unified flow, proposed herein for threshold logic synthesis, with the previous work presented in [25], being both based on cut pruning, as discussed in Section IV. Opencore circuits have been adopted as benchmarking [35]. Table I presents the results obtained. Columns 2-7 refer to the method described in [25], comprising the three steps illustrated in Fig. 1(b), and present the mapping results (number of TLGs and circuit logic depth) followed by the execution time of each stage. Columns 8-10 refer to the single-step mapping proposed in this work. The execution time reduction is up to 53%, being 36% on average, so demonstrating the advantage on identifying threshold functions during the cut enumeration task. The new threshold synthesis flow improves both the QoR and the execution time in terms of TLG count, without loss on circuit logic depth metric.

The benefits of performing threshold logic synthesis while relaxing the cut redundancy checking during the priority cuts enumeration is also demonstrated taking into account the opencore circuits as benchmarks [35]. Table II presents the results in terms of the number of TLGs and the circuit logic depth on the final threshold network, along with the runtime and memory usage during the execution. Columns 2-5 refer to the traditional cut enumeration, pruning the redundant cuts. Columns 6-9 refer to the proposed approach in which redundant cuts are enumerated during the threshold logic synthesis. The results show that the proposed strategy provided a circuit area reduction up to 21%, being 11% on average, with no significant impact on the circuit logic depth. Although there are runtime and memory overheads, none of the circuit synthesis has taken longer than 4 minutes nor needed more than 130 MB of memory usage during the mapping task.

It is expected that different area estimation criteria may deliver different topologies and network arrangements for the mapped circuits. Table III presents the comparison of the obtained results when the proposed mapper addresses the

Table II
TRADE-OFF OBTAINED WHEN RELAXING THE CUT REDUNDANCY CHECKING.

	Pruning Redundant Cuts				*Enumerating Redundant Cuts*			
circuit	*#TLGs*	*logic depth*	*run-time (sec)*	*memory (MB)*	*#TLGs (ratio)*	*logic depth (diff)*	*run-time (ratio)*	*memory (ratio)*
oc_aquarius	9426	29	20.55	20.62	(0.97)	0	(1.09)	(1.22)
oc_cfft_1024x12	4997	8	6.51	6.81	(0.79)	0	(1.70)	(3.46)
oc_cordic_p2r	4763	7	6.15	10.12	(0.83)	+1	(1.66)	(2.42)
oc_cordic_r2p	5752	7	6.95	10.27	(0.84)	+1	(1.63)	(2.32)
oc_des_perf	10415	7	26.59	49.39	(0.88)	0	(1.72)	(2.16)
oc_ethernet	3774	7	4.79	5.48	(0.98)	0	(1.11)	(1.15)
oc_fpu	9345	281	21.10	24.29	(0.87)	+1	(1.24)	(1.98)
oc_mem_ctrl	6642	9	7.44	3.70	(0.99)	-1	(1.08)	(1.57)
oc_video_dct	17075	14	35.76	51.22	(0.86)	-1	(1.22)	(2.17)
oc_video_jpeg	20965	13	35.62	49.80	(0.90)	-1	(1.25)	(2.26)
radar20	34254	14	54.90	44.63	(0.89)	0	(1.26)	(2.02)
uoft_raytracer	70263	23	168.61	87.34	(0.86)	0	(1.18)	(1.39)
geomean					(0.89)		(1.32)	(1.92)

Table III
TECHNOLOGY MAPPING OPTIMIZATION TARGETING DIFFERENT TLG AREA ESTIMATION.

	area estimation ($\sum$W + T)			area estimation ($\sum$#inputs)		
circuit	CF=$\sum$W + T	CF=#TLGs	(ratio)	CF=$\sum$#inputs	CF=#TLGs	(ratio)
oc_aquarius	64776	97130	(1.50)	28372	32908	(1.16)
oc_cfft_1024x12	22949	42464	(1.85)	12826	14662	(1.14)
oc_cordic_p2r	27761	39244	(1.41)	12370	13310	(1.08)
oc_cordic_r2p	32236	50198	(1.56)	15455	16733	(1.08)
oc_des_perf	58212	93687	(1.61)	29378	30307	(1.03)
oc_ethernet	23349	35979	(1.54)	11695	12974	(1.11)
oc_fpu	43445	58186	(1.34)	23374	24269	(1.04)
oc_mem_ctrl	34363	54483	(1.59)	19684	21156	(1.07)
oc_video_dct	98118	175844	(1.79)	45729	54351	(1.19)
oc_video_jpeg	123335	232028	(1.88)	60407	68898	(1.14)
radar20	161436	241578	(1.50)	86378	94799	(1.10)
uoft_raytracer	355222	593636	(1.67)	191468	210825	(1.10)
geomean			(1.60)			(1.10)

standard cost function (*i.e.*, the total number of TLGs) to other two area cost functions: the total sum of input weights and threshold values, and the gate fanin. Columns 2-4 present the total sum of input weights and threshold values obtained from the two cost functions. Columns 5-7 show the total number of gate inputs when targeting the evaluated cost functions. Notice that the best results have been obtained when the most appropriate area estimation is taken into account during the mapping process. Regarding the total sum of input weights and threshold values, the overhead has been up to 88%, being 60% on average, when minimizing the number of TLGs. Concerning the gate fanin, the results have been up to 19% worse, being 10% on average, whether the mapper optimizes the number of TLGs instead of the considered area cost function. Therefore, a threshold logic mapper capable of optimizing circuits taking into account different area estimations tends to be more effective when targeting different technologies due to their particularities.

B. Comparison to Other Approaches

In order to compare the proposed synthesis flow to other approaches, from different authors, three experiments were carried out. The first one compares the number of TLGs and the circuit logic depth to the results obtained from both strategies presented by Chen *et al.*, in [13], and by adopting a commercial tool. In the second one, the proposed approach is compared to the Gowda's method, in [9], and the Palaniswami's method, in [10], in terms of the number of TLGs. It is done because the most recent Chen's work does not compare itself to those approaches. In the third experiment, the circuit area results are compared in terms of the sum of input weights and threshold value to the results presented in [7] and in [12].

The comparison of our method to the Chen's approach was carried out taking into account the IWLS 2005 benchmark suite [36]. Table IV shows the obtained results in terms of TLG count and circuit logic depth. Notice that the Chen's method already provides improvement of 28% in TLG count and 14% in logic depth when compared to the Zhang's approach [7]. Therefore, the Cheng's method has been adopted as reference herein.

When limiting the TLGs to six inputs, the proposed method reduced the TLG count in 94% of the circuits, with reductions up to 39% of such a count, being 20% on average. The logic depth has been reduced in all applied benchmark circuits, with reductions up to 64% and being 53% on average. The runtime is less than one second per circuit synthesis.

In order to exploit the gate level scalability, we have synthesized circuits taking into account TLGs with up to 15 inputs. In this case, the TLG count has been reduced in all circuits, with reductions up to 47% and being 25% on average. The reduction in terms of logic depth has been up to 67%, being 57% on average. In the same experiment, we have also synthesized the circuits by adopting a commercial tool. To do that, we provided the tool with a cell library composed by all NPN threshold functions with up to six variables. Notice that, although the commercial tool improves the Cheng's results in terms of TLG count, our method has been able to improve these results even more. On average, the commercial tool improves 7% whereas the proposed approach improves 15%. Moreover, TLG count and circuit logic depth are simultaneously reduced by the proposed flow, whereas the commercial tool increases the Cheng's results in around 38% in terms of logic depth.

In [10], the authors present two different improvements, named BDD decomposition method (BDM) and ZDD decomposition method (ZDM), to the max literal factor tree (MLFT) method proposed by Gowda *et al.*, in [9]. The results shown in Fig. 4 present the TLG count obtained by these methods and by the one proposed herein. The ISCAS'85 set of benchmarks has been applied for this evaluation. When compared to the MLFT approach, BDM and ZDM methods provide an average TLG count reduction of 12% and 17%, respectively. The average reduction obtained by our method is about 65% and 48% when compared to MLFT and ZDM, respectively.

Finally, we compare our results to the work presented by Lin *et al.*, in [12], in terms of the summation of input weights and threshold value. This method starts from a TLG netlist (*i.e.*, a given TLN) generated by the Zhang's method, in [7], and performs a rewiring procedure, optimizing the TLG area cost function. Table V shows the results from this experiment. In [12], the Lin's method improves the Zhang's

Table IV
COMPARISON TO THE RESULTS PRESENTED BY CHEN [13] AND BY APPLYING A COMMERCIAL TOOL.

	number of TLGs				*circuit logic depth*			
circuit	*Chen* [13]	*Commercial Tool*	*Proposed K=6*	*Proposed K=15*	*Chen* [13]	*Commercial Tool*	*Proposed K=6*	*Proposed K=15*
spi	1614	(0.78)	(0.74)	(0.71)	19	(1.21)	(0.53)	(0.47)
systemcaes	5333	(0.82)	(0.79)	(0.77)	33	(0.88)	(0.42)	(0.42)
steppermotordrive	83	(0.75)	(0.71)	(0.61)	7	(2.57)	(0.43)	(0.43)
tv80	3559	(0.91)	(0.81)	(0.76)	30	(1.40)	(0.47)	(0.37)
ac97_ctrl	6194	(1.01)	(0.95)	(0.91)	7	(1.43)	(0.57)	(0.43)
sasc	333	(1.04)	(0.92)	(0.89)	7	(1.14)	(0.43)	(0.43)
pci_conf_cyc_addr_dec	62	(0.89)	(0.68)	(0.68)	4	(1.50)	(0.50)	(0.50)
usb_funct	6842	(0.93)	(0.82)	(0.78)	19	(1.42)	(0.53)	(0.47)
mem_ctrl	4721	(0.77)	(0.76)	(0.71)	23	(1.30)	(0.39)	(0.35)
systemcdes	1377	(0.90)	(0.82)	(0.77)	19	(1.16)	(0.47)	(0.47)
i2c	482	(0.87)	(0.76)	(0.69)	11	(1.45)	(0.36)	(0.36)
pci_bridge32	10497	(0.91)	(0.89)	(0.87)	21	(1.95)	(0.38)	(0.33)
aes_core	10057	(0.97)	(0.89)	(0.78)	17	(1.29)	(0.53)	(0.53)
simple_spi	436	(1.01)	(0.85)	(0.82)	8	(1.38)	(0.50)	(0.38)
des_area	2011	(1.01)	(1.01)	(0.95)	20	(1.40)	(0.55)	(0.50)
wb_conmax	21956	(0.87)	(0.82)	(0.80)	13	(1.62)	(0.62)	(0.54)
pci_spoci_ctrl	399	(0.64)	(0.61)	(0.53)	12	(1.50)	(0.42)	(0.33)
usb_phy	221	(0.85)	(0.70)	(0.64)	7	(1.14)	(0.43)	(0.43)
geomean		(0.88)	(0.80)	(0.75)		(1.39)	(0.47)	(0.43)

results for all benchmarks, obtaining a reduction of 4% on average. Our approach does not depend on a preliminary threshold synthesis and optimizes the cost function performing a threshold logic technology mapping directly over the original circuit description. Therefore, we have improved the Zhang's results for all benchmarks, so reducing the circuit area up to 46%, being 31% on average.

C. Analysis for Different Mapping Goals

The proposed threshold logic synthesis flow explores the LUT-based technology mapping strategy, which allows for near-optimum circuit logic depth covering [15]. From such a covering, three different mapping goals can be targeted in circuit area optimization. The first one chooses a cut that decreases the area even when increasing the logic depth (area oriented). The second one never replaces a cut if the logic depth is increased (delay oriented). Finally, an intermediate strategy chooses a cut that decreases the area whether the increasing in logic depth is less than a given pre-defined percentage (relaxed delay). In the experiments, we have allowed to increase the delay up to 30%.

In the following, we present a comprehensive set of experimental results when addressing the proposed mapper to the aforementioned goals and targeting different threshold-based circuit area estimations. Notice that we are varying two ternary possibilities in this experiment: the mapping goal, which can be area oriented, delay oriented or relaxed one; and the area estimations that can be considered as the number of TLGs, the summation of input weights and threshold values, or the overall gate fanin. This yields nine different solutions to each synthesized circuit.

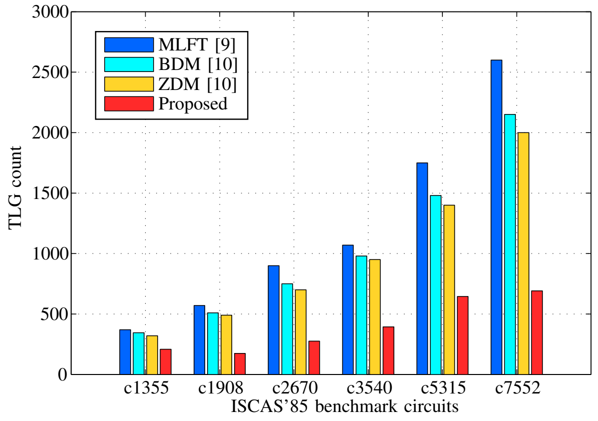

Figure 4. Comparison to the Palaniswamy's and Gowda's methods presented in [10] and in [9], respectively.

Table V
COMPARISON TO AREA RESULTS FROM ZHANG'S [7] AND LIN'S [12] APPROACHES.

	area estimation ($\sum W + T$) (ratio)				
circuit	*Zhang [7]*	*Lin [12]*		*Proposed*	
alu4	1986	1934	(0.97)	1973	(0.99)
apex6	2079	2007	(0.97)	1720	(0.83)
C1355	2102	2098	(1.00)	1312	(0.62)
C1908	1671	1631	(0.98)	1157	(0.69)
C5315	4661	4651	(1.00)	4133	(0.89)
C6288	9892	9844	(1.00)	7147	(0.72)
C7552	6468	6412	(0.99)	3972	(0.61)
dalu	3644	3608	(0.99)	2456	(0.67)
frg2	3299	2977	(0.90)	1928	(0.58)
i10	7490	6888	(0.92)	5472	(0.73)
i2c	3268	2867	(0.88)	2043	(0.63)
pair	4057	3945	(0.97)	3557	(0.88)
pci_spoci_ctrl	3254	3127	(0.96)	1472	(0.45)
rot	1960	1878	(0.96)	1550	(0.79)
s13207	9542	9221	(0.97)	5438	(0.57)
s9234	7056	6415	(0.91)	4149	(0.59)
simple_spi	2626	2540	(0.97)	2085	(0.79)
spi	12004	11184	(0.93)	6523	(0.54)
systemcdes	11677	11139	(0.95)	7190	(0.62)
usb_phy	1586	1498	(0.94)	1176	(0.74)
x3	2170	2054	(0.95)	1730	(0.80)
Geomean			(0.96)		(0.69)

Table VI presents the results related to the circuit area and logic depth. In columns 2-7, the area is estimated in terms of TLG count. In columns 8-13, the area is estimated in terms of the summation of input weights and threshold values. In columns 14-19, the area is estimated in terms of the overall number of TLG inputs. Column 20 presents the worst case runtime, in seconds, taking into account all the nine synthesis performed in this experiment. We have adopted different benchmark suites, such as the EPFL more-than-million (MTM), arithmetic and random-control [37], ISCAS'85 [36] and the opencore circuits [35]. The circuit level scalability has been verified by synthesizing benchmarks comprising more than 20 million AIG nodes.

VI. Conclusions

In this paper, an effective logic synthesis for circuits based on threshold logic gates (TLGs) is presented. The main contributions of this work are the following: (1) an efficient threshold logic synthesis flow based on cut pruning, which reduces area and delay, being also scalable to large benchmark circuits; (2) a clever way to explore redundant cuts in order to improve the quality of results; and (3) a technology mapping able to handle different TLG area estimations: the total sum of input weights and threshold values, the gate fanin and the total number of TLGs. When compared to the state-of-the-art related methods, the proposed approach have reduced up to 47% the circuit delay. Finally, we also presented experimental results by customizing the developed mapper to achieve distinct mapping objectives (area x delay) and targeting different threshold-based area estimations. The proposed threshold logic synthesis is available in the public ABC tool.

References

[1] L. Gao, F. Alibart, and D. B. Strukov, "Programmable cmos/memristor threshold logic," *IEEE Trans. on Nanotechnology*, vol. 12, no. 2, 2013.

[2] 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.

[3] S. Vrudhula, N. Kulkami, and J. Yang, "Design of threshold logic gates using emerging devices," in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, 2015, pp. 373–376.

[4] N. S. Nukala, N. Kulkarni, and S. Vrudhula, "Spintronic threshold logic array (stla) - a compact, low leakage, non-volatile gate array architecture," in *Proc. of Int'l Symp. on Nanoscale Architectures*, 2012.

[5] Z. He and D. Fan, "Energy efficient reconfigurable threshold logic circuit with spintronic devices," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 223–237, 2017.

[6] H. Pettenghi, M. J. Avedillo, and J. M. Quintana, "Improved nanopipelined rtd adder using generalized threshold gates," *IEEE Trans. on Nanotechnology*, vol. 10, no. 1, pp. 155–162, 2011.

[7] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits Syst.*, vol. 24, no. 1, 2005.

[8] J. L. Subirats, J. M. Jerez, and L. Franco, "A new decomposition algorithm for threshold synthesis and generalization of boolean functions," *IEEE Trans. on Circuits Syst. I*, vol. 55, no. 10, 2008.

[9] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits Syst.*, vol. 30, no. 5, 2011.

[10] A. K. Palaniswamy and S. Tragoudas, "Improved threshold logic synthesis using implicant-implicit algorithms," *ACM Journal on Emerg. Tech.*, vol. 10, no. 3, 2014.

[11] A. Neutzling, M. G. Martins, R. P. Ribas, A. Reis, *et al.*, "A constructive approach for threshold logic circuit synthesis," in *Proc. of Int'l Symp. on Circuits and Syst.*, 2014.

[12] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for threshold logic circuit minimization," in *Proc. of Conf. on Design, Automation & Test in Europe*, 2014.

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

[14] Berkeley Logic Synthesis and Verification Group, "Abc: a system for sequential synthesis and verification," Release 20130425. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/.

[15] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to technology mapping for lut-based fpgas," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits Syst.*, vol. 26, no. 2, 2007.

[16] 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.

[17] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," in *Proceedings of the 35th annual Design Automation Conference*, ACM, 1998, pp. 206–211.

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

[19] A. Neutzling, M. G. Martins, V. Callegaro, R. P. Ribas, and A. Reis, "A simple and effective heuristic method for threshold logic identification," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits Syst.*, to be published.

[20] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, "A generalized approach to implement efficient cmos-based threshold logic functions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–14, 2017. DOI: 10.1109/TCSI.2017.2768563.

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

[22] D. Chen and J. Cong, "Daomap: a depth-optimal area optimization mapping algorithm for fpga designs," in *Int'l Conf. on Comput.-Aided Design (ICCAD)*, 2004.

[23] J. Cong and Y. Ding, "Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits Syst.*, vol. 13, 1994.

[24] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *Int'l Conf. on Comput.-Aided Design*, 2007.

[25] A. Neutzling, J. M. Matos, A. I. Reis, R. P. Ribas, and A. Mishchenko, "Threshold logic synthesis based on cut pruning," in *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2015.

[26] M. G. Martins, R. P. Ribas, and A. I. Reis, "Functional composition: A new paradigm for performing logic synthesis," in *Quality Electronic Design (ISQED), 2012 13th Int'l Symp. on*, IEEE, 2012, pp. 236–242.

[27] A. Neutzling, M. G. Martins, R. P. Ribas, and A. I. Reis, "Synthesis of threshold logic gates to nanoelectronics," in *Integrated Circuits and Systems Design (SBCCI), 2013 26th Symposium on*, IEEE, 2013, pp. 1–6.

[28] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold circuits," in *Proc. of Int'l Conf. on Comput.-Aided Design.*, 2011.

[29] V. Annampedu and M. D. Wagh, "Decomposition of threshold functions into bounded fan-in threshold functions," *Information and Computation.*, vol. 227, pp. 84–101, 2013.

[30] N. Kulkarni, J. Yang, J.-S. Seo, and S. Vrudhula, "Reducing power, leakage, and area of standard-cell asics using threshold logic flip-flops," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2873–2886, 2016.

[31] A. Neutzling, J. M. A. Maick, A. I. Reis, and R. P. Ribas, "Technical report: Number of prime implicants in unate functions," 2018. [Online]. Available: http://www.inf.ufrgs.br/logics/number_of_primes.

[32] N. Kulkarni and S. Vrudhula, "Efficient enumeration of unidirectional cuts for technology mapping of boolean networks," *ArXiv preprint arXiv:1603.07371*, 2016.

[33] D. Kagaris and S. Tragoudas, "Maximum weighted independent sets on transitive graphs and applications1," *Integration, the VLSI journal*, vol. 27, no. 1, pp. 77–86, 1999.

[34] A. Neutzling, J. M. Matos, A. Mishchenko, A. I. Reis, and R. P. Ribas, *Dataset for: Effective logic synthesis flow for threshold logic circuit design*, 2017. [Online]. Available: http://dx.doi.org/10.17632/ypvc8p99cb.1.

[35] J. Pistorius, M. Hutton, A. Mishchenko, and R. Brayton, "Benchmarking method and designs targeting logic synthesis for fpgas," in *Proc. of Int'l Workshop on Logic and Synthesis*, vol. 7, 2007.

[36] C. Albrecht, "Iwls 2005 benchmarks," in *Int'l Workshop on Logic and Synthesis (IWLS)*, 2005.

[37] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Int'l Workshop on Logic and Synthesis (IWLS)*, 2015.

Augusto Neutzling (S'13) received the B.S. degree in Computer Engineering from Federal University of Rio Grande (FURG), Rio Grande, Brazil, in 2012, the M.S. and Ph.D. degree in Computer Science from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2014 and 2017 respectively. He also received his MBA in Project Management at the Getulio Vargas Foundation(FGV), in 2014. He was a trainee in AMS and digital integrated circuit design at IC Brazil Program from the Brazilian government.. His research interests are electronic design automation (EDA), logic synthesis methods for CMOS and emerging technologies, threshold logic, and cell library design.

Jody Maick Matos (S'12–M'18) received the B.S. degree in computer engineering from the State University of Feira de Santana, Brazil, in 2013, the M.Sc. degree in microelectronics and the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Brazil, in 2014 and 2018, respectively. His current research interests include data structures and algorithms for the VLSI design flow. Dr. Matos was a recipient of the A. Richard Newton Young Student Fellow Award in 2017, the First Place Prize at the IWLS 2017 Programming Contest, and the IWLS 2015 Best Student Paper Award.

Alan Mishchenko graduated from Moscow Institute of Physics and Technology (Moscow, Russia) in 1993 with MS and received his PhD from the Glushkov Institute of Cybernetics (Kiev, Ukraine) in 1997. From 1998 to 2002 he was an Intel-sponsored visiting scientist at Portland State University. Since 2002, he has been a professional researcher in the EECS Department at UC Berkeley. Dr. Mishchenko shared the D.O. Pederson TCAD Best Paper Award in 2008 and the SRC Technical Excellence Award in 2011 for work on ABC. His research interests are in developing computationally efficient methods for synthesis and verification.

Andre Reis (M'99-SM'05) received the B.S. degree in Electrical Engineering from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1991, the M.S. degree in Computer Science also from UFRGS, in 1993, and the Ph.D. degree in Automatic and Microelectronics Systems from UMII, Montpellier, France, in 1998. He is currently professor in the Department of Applied Informatics, at Institute of Informatics, UFRGS, since 2000. He was a visiting researcher at University of Minnesota, Minneapolis, USA, in 2004-2005.

Renato P. Ribas (M'12) received the B.S. degree in Electrical Engineering from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1991, the M.S. degree in Electrical Engineering from University of Campinas (Unicamp), Campinas, Brazil, in 1994, and the Ph.D. degree in Microelectronics from Institut National Polytechnique de Grenoble, France, in 1998. He is currently professor in the Department of Applied Informatics, at Institute of Informatics, UFRGS, since 2000.

Table VI
SUMMARY OF EXPERIMENTAL RESULTS OBTAINED FOR ALL DESIGN COST FUNCTIONS AND AREA ESTIMATIONS.

	area = #TLGs						area = $\sum$W + T						area = $\sum$#Inputs						
	area oriented		level oriented		relaxed level		area oriented		level oriented		relaxed level		area oriented		level oriented		relaxed level		worst case
circuit	area	level	area	level	area	level	area	level	area	level	area	level	area	level	area	level	area	level	runtime
C1355	252	13	211	7	208	9	1312	13	2091	7	1648	9	656	13	908	8	868	10	0.73
C1908	175	18	192	10	174	13	1162	21	1637	10	1532	13	570	21	683	11	610	14	1.33
C2670	271	14	281	7	276	10	1777	13	2231	7	1955	9	828	15	869	10	835	13	1.22
C3540	396	24	398	13	394	16	2244	30	3155	13	2922	16	1361	25	1562	13	1396	18	2.55
C5315	650	18	669	10	645	11	4106	19	5648	9	4830	13	1993	18	2162	11	2052	14	4.4
C7552	678	32	714	10	692	11	3991	36	5645	10	4769	13	2150	30	2326	11	2240	14	6.1
oc_aquarius	8990	84	9179	29	9095	37	64776	92	83105	29	78035	37	28372	89	30406	32	29970	44	51
oc_cfft_1024x12	3943	22	3963	8	3961	10	22949	25	32365	8	30096	10	12826	26	14500	9	14170	11	24
oc_cordic_p2r	3787	20	3936	8	3886	9	27761	21	34454	8	32378	10	12370	22	13829	9	13331	11	22
oc_cordic_r2p	4720	22	4841	8	4794	9	32236	25	43353	8	41821	10	15455	27	17433	8	16998	10	24
oc_des_perf	8405	12	9136	7	8521	9	58212	13	84235	7	65235	9	29378	12	35223	7	31361	9	94
oc_ethernet	3648	24	3689	7	3651	9	23349	33	26493	7	25266	9	11695	33	12246	9	12072	11	12
oc_fpu	7816	948	8173	282	7973	360	43445	1016	79646	388	70018	361	23374	997	27616	291	26706	384	58
oc_mem_ctrl	6521	16	6588	8	6548	10	34363	25	41943	8	40089	10	19684	21	20317	11	19907	14	19
oc_video_dct	14003	29	14659	13	14290	16	98118	34	157479	13	149972	16	45729	31	55449	14	52370	18	85
oc_video_jpeg	18128	30	18879	12	18632	15	123335	34	171704	12	158158	15	60407	33	67014	14	65289	18	90
radar20	28782	45	30346	14	29719	18	161436	47	258722	14	229448	18	86378	45	98597	17	96131	19	126
uoft_raytracer	59232	70	60626	23	59925	29	355222	88	522004	23	481115	29	191468	85	211060	25	206744	32	360
adder	286	102	341	33	322	42	1967	129	3212	33	3069	42	906	129	1112	33	1095	42	2
bar	1792	14	1663	5	1406	7	8960	14	17856	5	13306	7	4480	14	5310	8	4941	10	4
div	17374	2117	20001	568	16392	752	85465	2272	144716	502	129789	705	47575	2200	43028	585	60450	790	120
hyp	129857	7337	137089	3189	134277	4167	731970	8778	1026321	3208	919501	4170	362848	8660	410824	3284	395135	4279	1501
log2	16236	218	15570	91	15908	118	94205	221	142949	91	128127	119	46908	222	57093	97	52602	128	127
max	954	82	1398	35	1051	61	7089	179	16106	36	13199	49	3040	98	3891	56	3207	70	4
multiplier	15458	170	15279	62	15303	80	88022	179	119674	62	99787	78	41265	176	47965	64	43516	84	68
sin	2687	108	2690	43	2564	57	12844	141	28328	42	25839	54	7911	129	10505	46	10187	65	26
sqrt	11232	2169	12748	879	12585	1160	52842	2306	131820	885	121237	1160	31598	2167	42689	625	41867	803	70
square	9898	118	10007	42	9911	54	56593	132	67512	42	59345	49	28680	123	30738	50	28818	66	57
arbiter	2501	19	2065	13	1639	14	35238	45	141350	13	98878	16	14090	15	14235	14	14090	18	28
cavlc	189	7	198	4	194	5	1584	9	2602	4	1983	5	806	7	823	6	816	6	1
ctrl	40	3	41	2	41	2	326	5	434	2	434	2	148	5	168	2	168	2	0.1
dec	256	1	256	1	256	1	912	3	4096	1	4096	1	608	3	2048	1	2048	1	0.23
i2c	348	7	354	4	352	5	2204	12	2789	4	2416	5	1297	10	1384	4	1378	5	1
int2float	60	6	59	4	60	5	543	9	723	4	598	5	265	6	269	5	267	6	0.28
mem_ctrl	8739	32	9088	10	8909	13	59568	42	100377	10	87454	13	32331	36	35080	12	34439	15	64
priority	87	18	87	17	85	23	1181	60	2536	17	2263	22	532	19	536	17	532	23	0.32
router	29	6	31	4	30	5	382	10	615	4	492	5	171	8	189	4	182	5	0.18
voter	4229	41	6241	24	4829	32	24338	48	58510	24	28284	32	12692	47	22230	28	13110	33	
sixteen	4377156	50	4393330	31	4377179	40	5464453	87	5183416	31	5425647	40	4459711	53	4470871	34	4459728	44	1392
twenty	5479332	54	5500745	34	5479282	44	7458200	69	7420364	34	8262487	44	5675698	53	5678591	39	5675698	52	1983
twentythree	6127154	52	6148970	38	6126980	49	8541690	108	7812179	38	8395822	49	6364909	51	6369649	42	6364909	51	2489