An Input Weights Aware Synthesis Tool for Threshold Logic Networks

Li Zhang and Sorin Cotofana
Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands
{zhangli, sorin}@CE.ET.TUDelft.NL

Abstract— **In this paper we present a TL specific synthesis tool that aims to exploit the specific characteristics of TL gates, especially the possibility to assign weights larger than one to TL gate inputs. Moreover, instead of confining network nodes with input fan-in restriction, we employ the sum of weights as constraint, as this is more relevant to the physical basis of TL implementations. To make a seamless concatenation with Boolean logic network access and synthesis, we embedded our program into a widely applied existing logic synthesis tool, SIS from University of Berkeley, by supplementing it with 8 new TL specific commands. Several efficient heuristic synthesis methodologies are proposed, tested, and implemented, and the user can change the synthesis approach by simply specifying the required heuristic. To evaluate the proposed tool we used 22 MCNC benchmark circuits. Our experiments suggest that when compared with the state of the art our system can achieve, on average, a 24.5% node count reduction (area), a 32.8% network depth reduction (delay), and a 49.4% product of node count and network depth (area delay product) reduction. Furthermore, our tool is able to provide well-balance networks, which is highly important for practical designs.**

Keywords— **Threshold Logic, Logic Synthesis, Integer Linear Programming**

I. Introduction

Threshold Logic (TL), as a powerful alternative to Boolean Logic (BL), was proposed in the late 1950's as a result of the development of neural network theory, and was proven in theory to be more efficient than Boolean Logic [1]. Recently, TL gates have been proposed in CMOS technology, e.g., Capacitive Threshold Logic [2], Differential Current-Switch Threshold Logic [3], and emerging nano-technologies, e.g., Single Electron Tunneling (SET) [4], Resonant Tunneling Diodes (RTD) [5], etc. While important steps have been made towards the hardware implementation of TL computing units the TL utilization in Very Large Scale Integration (VLSI) also requires the existence of high-level TL synthesis tools. Up to date very little has been done in this research direction [6]. Recently, a synthesis tool has been proposed in [7], but it mostly treats Threshold Logic Gates (TLG) as conventional BL gates and the particularities of TL gates are almost completely ignored during the synthesis process.

In the view of the previously mentioned facts we focus in this paper on effective TL network synthesis. More in particular, we concentrate on the efficient utilization of TL gate features, e.g., the capability to have input weight values larger than one in an Integrated Circuit (IC) implementation. Furthermore, we choose to embed our TL synthesis tool into SIS [8], a Boolean logic synthesis tool for synthesis and optimization of sequential and combinational circuits, to make a seamless concatenation with Boolean Logic network access and, at the same time, take advantage of SIS's powerful Boolean network optimization capability.

This paper is organized as follows. Section II introduces Threshold Logic and discusses state of the art VLSI implementations of TL. Additionally, the utilization of gate sum of input weights instead of gate fan-in is proposed as constraint for TL synthesis. Section III presents the basic synthesis procedure we implemented. Section IV presents in details the TL synthesis methodology. Various heuristic algorithms are presented. Section V presents the experimental results and comparisons among our various algorithms and with the approach in [7]. Section VI presents a summary of our work and some conclusions.

II. Background

A Threshold Logic Gate, or a threshold gate, is a logic gate assuming Boolean inputs and operating such that the output is 1 if the sum of the input quantities $x_1, x_2, ..., x_n$, weighted with real numbers $w_1, w_2, ..., w_n$, attains the threshold (t) (which is also a real number) and the output is 0 if the sum does not. This can be formulated as following:

$$F(X) = sgn\{f(X)\} = \begin{cases} 1 & \text{if} \quad f(X) \geq t \\ 0 & \text{if} \quad f(X) < t \end{cases} \tag{1}$$

$$f(X) = \sum_{i=1}^{n} w_i x_i - t \tag{2}$$

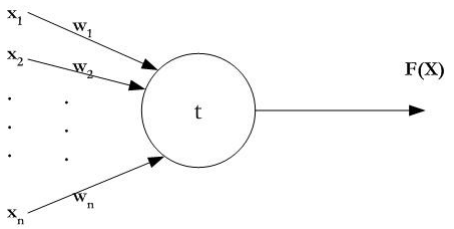

Fig. 1. A Threshold Logic Gate

Figure 1 depicts such a threshold logic gate. For a given TL gate with inputs $x_1, x_2, ..., x_n$, and each input having a weight of $w_1, w_2, ..., w_n$, respectively, in case the threshold for the gate is t, for the sake of notation simplicity we denote this gate as $[w_1, w_2, ..., w_n;\ t]$ with respect to $x_1, x_2, ..., x_n$, or as $[(x_1)w_1, (x_2)w_2, ..., (x_n)w_n;\ t]$.

TLGs are capable to efficiently realize complex Boolean functions using fewer logic gates [9]. For example, a Boolean function $f = ab + bc$ can be implemented into

a single TLG instead of 2 logic gates with Boolean gates. Due to its powerful capabilities in implementing Boolean functions, a number of VLSI implementations are built up to further explore the advantages of TLG. There are three main state of the art TL implementations: Capacitive Threshold Logic (CTL), conductance/current mode TL implementations, and implementations based on the emerging nano-technologies.

Previous research in TL synthesis [7] is using the gate fan-in as a synthesis constaint. We believe that fan-in is not the most appropriate metric for TL synthesis as it does not accurately reflect the physical phenomena that take place in a TLG implementation. In order to demonstrate that this is the case and to determine the most appropriate metric for TL synthesis, we use Differential Current-Switch TL (DCSTL) [3] as a discussion vehicle. DCSTL belongs to the conductance/current mode TL implementation family. As depicted in Figure 2, the circuit comprises a fast semi-dynamic CMOS latch and two analog computation blocks referred as *data bank* and *threshold mapping bank*. Both data and threshold mapping banks consist of two parallel-connected sets of unit nMOS transistors. In the precharge phase, both Y_1 and $\overline{Y_1}$ are precharged high. In the evaluation phase, the total currents generated by the transistor banks I_{data} and I_T, corresponding to the data bank and the threshold mapping bank, are compared with each other by the latched comparator. Both Y_1 and $\overline{Y_1}$ start dropping at different speeds affected by I_{data} and I_T. The node which crosses the latch switching threshold decides the output voltage of the TL gate. The input weights of the TLG are represented via the data bank transistors' geometric properties, and the threshold is represented by the threshold mapping bank transistors' geometric properties and the threshold mapping input values.

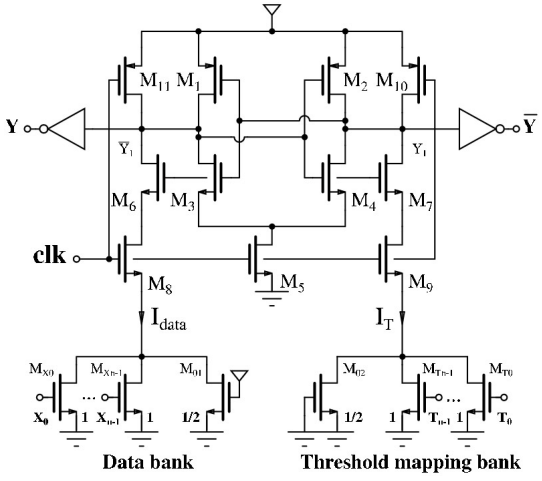

Fig. 2. A Differential Current-Switch Threshold Logic Gate

In Figure 2, the comparison between the sum of weighted inputs and the threshold is actually carried out on *current* level, and the currents that can be generated from the data and threshold banks decide the output of the circuit. Since different transistors in each bank represent different weights and threshold values, the voltage drops at Y_1 and $\overline{Y_1}$ are linear to the currents that can be drained from threshold and data banks, respectively. Thus, the voltage comparison in $\overline{Y_1}$ and Y_1 can be redefined as the comparison on the voltage drop on both points, which is $V_{dw} = V_{du} \times \sum_{i=1}^{n} w_i$ and $V_{dt} = V_{du} \times t$, where V_{du} represent the unit voltage drop. Since in extreme conditions, V_{dw} cannot be larger than V_{dd}, V_{du} cannot be higher than $V_{dd}/\sum_{i=1}^{n} w_i$. This actually means that in order to make the circuit able to be sensitive to a unit voltage difference, the sum of weights has to be confined to a certain range. Thus, in a TL tailored synthesis process, we should use the Sum Of Weights (SOW) as a constraint instead of the fan-in. Actually, SOW can be seen as the **generalized fan-in** metric, as when all the weights are 1's, it reduces to fan-in. We note here that the fact that fan-in is not the most appropriate metric for TL synthesis that holds fine in all TL implementation classes. Instead, SOW is the metric most correlated with the underlying physical structure of TLGs.

III. Threshold Logic Network Synthesis Overview

The overall organization of the TL synthesis framework we propose is depicted in Figure 3. Our proposal is based on two main steps. The first step (the preprocessing) is dedicated to the construction of a BL gate mapping for a given network specification. As we decided to embed our proposal in the SIS environment this can be taken care by SIS, thus, no new dedicated modules are required for the preprocessing. The second step is TL specific and constitutes the augmentation we did to the SIS environment. In the remainder of this section, we shortly describe the basic blocks in Figure 3.

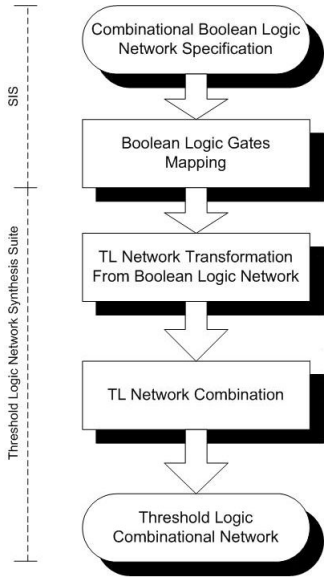

Fig. 3. Threshold Logic Synthesis Flow

First of all, the TL synthesis tool has to deal with traditional combinational Boolean logic (BL) network specifications. Most combinational circuits utilized in SIS are

described in BLIF (Berkeley Logic Interchange Format). Thus, in the synthesis process, BLIF has to be translated into Boolean network structures, usable to and understandable by the TL synthesis tool. An original SIS command, *read_blif*, is capable of doing this translation.

Since SIS is equipped with many powerful operations and algorithms for BL minimization and BL network mapping, which can optimize BL networks on both metrics of area and delay, we take advantage of SIS to apply a Boolean mapping on network specifications to get a pre-optimized network. This process is also done by original SIS commands.

After this stage, our TL network synthesis tool takes the control and starts a direct mapping to a TL network from a BL network. This process is called **TL Network Transformation**. After this stage, an equivalent TL network is generated on the basis of a BL network, and each basic NOT, AND, and OR gate is translated into the corresponding TL gate.

Here we present a simple example of direct mapping operation in Figure 4. On the graph, nodes and links are corresponding to the TL gates and wires between gates in the circuit, respectively. For instance, node n_1 represents a TL AND gate. Node a and b are corresponding to the primary inputs of the circuits. Node n_0 and n_1 are called sub-nodes of n_2. In the tree graph, primary inputs are represented as leaves, primary outputs are represented as roots.

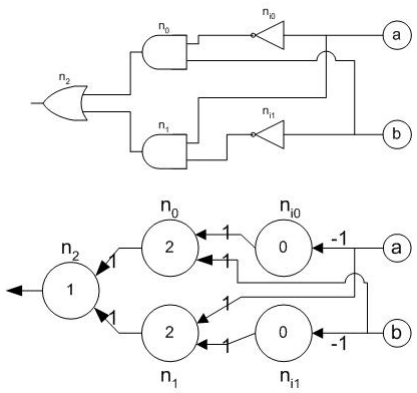

Fig. 4. A 2-input XOR Gate Direct Mapping

The directly mapped TL network has only simple TL gates replicating Boolean AND, OR, and NOT gates. To fully exploit the effectiveness of TL gates, a combination process has to be applied on the available TL network. At this stage, some simple TL gates can be combined together to create much more complex TL gates, which can perform the same functions. Totally, four combination algorithms (heuristics) are proposed to perform the TL gate combination. They are all implemented in the **TL Network Combination** module that is used to optimize the directly mapped TL network.

The TL network synthesis suite we propose is aiming the TL network optimization on metrics like circuit area, delay, and area delay product. Based on the discussion in Section II, the area of a TL gate depends on the sum of its input weights. For instance, the area of a DSCTL gate is composed out of the latch area (a constant area) and the area of the transistors in the data and threshold mapping banks. Thus, the area σ of a DSCTL gate can be expressed as $\sigma = \sigma_L + a \times SOW$, where σ_L is the latch area, a is a constant, and SOW is the sum of the input weights and threshold values. In this paper, for the sake of simplicity, we assume that all TL gates have the same unit area. We made this assumption mainly in order to easily compare our results with the results in previous TL synthesis researches [7], where the same assumption for TL gates was utilized. We note here however that some precise TL gate implementation specific area models can be easily included in our framework.

Within certain limits, the propagation delay of TL gates increases slowly when the gate fan-in increases. Compared with Boolean gates, TL gate propagation delay dependence with the fan-in number is logarithmic instead of linear or a higher order [10]. Thus, for that reason and for the sake of simplicity, we assume that each TL gate has a constant delay. Holding these two assumptions, the metrics of area, delay, and area delay product, are converted into the metrics of node count, network depth, and product of node count and number of network levels. Later on, the two set of metrics are regarded as interchangeable in this paper.

IV. Threshold Logic Network Combination

While TLGs are capable of implementing complex Boolean functions, the TL network achieved from the direct mapping stage is still using TL gates implementing basic Boolean operations. To exploit the powerful capabilities of TLGs, TL node combinations have to be carried out to increase the nodes' computation complexities and reduce the total number of nodes and the number of levels in the network. However not all the Boolean functions can be expressed as threshold functions, which means that the combination has to stop under some certain circumstances. To find out whether or not a certain Boolean function can be expressed as a TL function, Integer Linear Programming (ILP) technique is utilized. Thus before introducing the combination heuristics we first present the ILP formulation methods, which are used to map Boolean functions into TL functions. Before explaining the method, several terms has to be clarified. A function f is *positive (negative) unate* in variable x_i if and only if f is monotone increasing (decreasing) in x_i, otherwise f is a *binate* function [11]. If f is positive (negative) unate in all its variables, f is a *positive (negative) unate function* [11]. If f is *unate* in all its variables, regardless of the fact that it is positive or negative, f is a *unate function*. The *ON-set* of function is the set of input vectors that make the function output to be 1. Similarly, the *OFF-set* of function is the set of input vectors that make the function output to be 0.

Since a binate function cannot be a threshold function [12], the Sum Of Products (SOP) of the Boolean function should also be checked for its unateness. However, not all the unate funcions are threshold functions. For instance,

$f = ab + cd$ cannot be threshold function. Actually, only if the ILP problem formulated from the SOP is solvable, can the function be realized with a TLG.

In [6], ILP was proposed to determine the input weights and threshold of a TL gate. A series of relationships between the weights and the ON-set and OFF-set of a Boolean function was developed.

The traditional algorithm can only handle positive unate functions, thus some literal substitution has to be performed to make the unate function positive unate. For example, consider a unate function f, where $f = a_1 \vee \bar{a}_2 a_3$. In this case a_2 is in inverted (negative) form. If we replace $\bar{a}_2$ with b_2, we get a positive unate function $f = a_1 \vee b_2 a_3$, where $b_2 = \bar{a}_2$.

Then, the ILP formulation for f with respect to a_1, b_2, a_3 and threshold t is as follows:

Minimize: $w_1 + w_2 + w_3$
Subject to: $w_i \geq 0$, for $i = 1, 2, 3$.
$t \in$ integer.

ON set: $a_1 \vee b_2 a_3$
$w_1 + 0 + 0 - t \geq 0$, for cube a_1
$0 + w_2 + w_3 - t \geq 0$, for cube $b_2 a_3$

OFF set: $\bar{a_1}\bar{b_2} \vee \bar{a_1}\bar{a_3}$
$0 + 0 + w_3 - t \leq -1$, for cube $\bar{a_1}\bar{b_2}$
$0 + w_2 + 0 - t \leq -1$, for cube $\bar{a_1}\bar{a_3}$

The detailed formulation method and proof of this formulation method can be found in [9].

A. TL Non-Stop Recursive Constrained Combination

The basic idea of the Non-Stop recursive combination method (Non-Stop method) is to recursively combine all the sub-branches, whenever possible. For instance, in Figure 4 the tree led by node n_1 is the sub-branche of the tree led by node $n2$.

Before applying the method, the whole TL network G consists of purely AND, OR, and/or NOT gates in TL version. This Non-Stop method takes the sub-network from each primary output node, po_i, and supplies each primary output to the recursive Non-Stop combination procedure. In the procedure, the Sum Of Products (SOP) of the current node is constructed. For instance, in Figure 4 the SOP representation of node n_2 is $n_0 + n_1$. Each sub-node n_i of the node n is tested for combinability into the top node n. This test is carried out at the Boolean sum of products level. In Figure 4, node n_0 is first tested for the posibility to be absorbed into node n_2. After replacing the literal n_i with the function it represents into n's SOP, the newly created SOP is passed to a TL gate feasibility procedure which employs ILP. In Figure 4, the new SOP of node n_2 is $n_{i0}b + n_1$. If the SOP passes the TL feasibility procedure and satisfies SOW constraint, the newly created SOP will replace the current one, otherwise the current SOP is kept on trail with next sub-nodes. This loop keeps running until all the sub-nodes are tested once. In Figure 4, since the new SOP is a threshold function which satisfies SOW constraint, the SOP on stack is updated. The other sub-node n_1 will be tried next in the same way. However, in this case the new SOP cannot be a threshold function, thus $n_{i0}b + n_1$ is still the SOP kept on stack.

After all the sub-nodes are processed and the SOP is updated, if any sub-nodes were regarded as absorbable to the top node n during the process, a node combination occurs on node n and a new node n' is created from SOP to replace the original node n. In Figure 4, the new TL node n'_2, $[(n_{i0})1, (b)1, (n_1)2; 2]$, can be created from $n_{i0}b + n_1$. Then node n' is supplied to the Non-Stop procedure again. In case no branch of node n was successfully absorbed into SOP, the node is then marked in order to signal that it cannot be combined and each of the sub-nodes, n_i, are supplied into the recursive combination procedure.

Since some fan-out nodes provide signals for many successors, which can be regarded as sharing nodes in the network, the absorption on these nodes might lead to the duplicating of some components. Basically, the Non-Stop method is aiming to combine as many nodes as possible regardless the original node sharing properties in the network. Also due to the simple traversal sequence, some nodes may be left unprocessed in the network.

B. Node-By-Node Level Priority Combination

The second heuristic is aiming to improve the performance of the combination method by altering the traversing sequence. Unlike the Non-Stop method, which can be regarded as a partially Depth-First Traversal, Node-by-Node Level Priority Recursive Combination is more likely to be a Breadth-First Traversal.

Instead of traversing sub-nodes after finishing the absorption on the current node, the procedure can return immediately and try next node on the same level. The procedure starts from the root level (primary outputs) in a network. After passing through the top level, the traversing algorithm goes toward primary inputs downward in a level-by-level fashion. Due to the accessing pattern of this approach, we name this procedure Recursive Node-by-Node Level Priority Combination method (LP method). This strategy prevents the system from processing lower level nodes before higher level nodes, and consequently can avoid the situation when some nodes are left unprocessed.

C. Bottom-up Analysis Based Combination

In the first two algorithms, we tried to carry out as many combinations as possible, while sacrificing some node sharing properties. In this method, we proposed an alternative approach, which is based on traversing the network from primary inputs to primary outputs. From the bottom, the system gets all the nodes in the current, which is the primary input level in the beginning, sort inputs for each node in this level and try to combine all the nodes upon the knowledge collected during an analysis step. Since after the combination procedure on a level the network topology might change, with regard to the original depth properties marked on each nodes, the procedure might get incorrect nodes for a certain level. In order to let the procedure al-

ways have access to the correct node list for a certain level, the level property of combined nodes should always be updated in time. Thus, this algorithm is called Bottom-up Analysis Based Combination method (BAB method).

The real node absorption (combination) is carried out upon the information gathered in an analysis step. For each level, the combination is carried out in two phases, analysis and absorption. Our strategy is trying to combine the nodes that can only be absorbed by all their fan-out nodes. In case a node can only be absorbed by part of its fan-out nodes, this node is not absorbed by any of its successors. In this way, the node sharing properties in the network are well preserved.

D. Dual Direction 2-pass Analysis Based Combination

The first two combination algorithms are targeting on more combination, and the third one is trying to preserve the node sharing property in the network. Thus we propose another algorithm and try to balance the two factors in the combination process.

This new method is a two-pass method. It first applies an analysis on the entire network, and implements the real combination based on the analysis result. Unlike BAB method, in which the analysis and combination are performed for each level, both operations of the new method are applied to the whole network. The analysis phase follows an algorithm similar to the LP approach from the top of the network, primary outputs, to the bottom, primary inputs. The combination phase combines the network in a bottom-up fashion. Thus, we name the algorithm Dual Direction 2-pass Analysis Based Combination method (DDAB method).

Although it is still an heuristic thus it does not produce optimal solutions, it gives stable performance on the networks we used for its evaluation and provides balance on the node sharing and node absorption. Due to this property, the DDAB combination method is set as the default option for the ILP based combination in our TL synthesis tool.

V. Experimental Results

To evaluate our proposal we conducted a number of experiments on a subset of the MCNC benchmarks [13]. In order to be able to compare our proposal with previous art, we utilize the same set of 22 benchmarks as in [7]. The major comparison metrics we consider are node count, number of levels in network, and the product of node count and number of levels, which are corresponding to the area, delay, and area delay product.

To evaluate the efficiency of the proposed algorithms, the result achieved from direct mapping and our heuristics are compared. As presented in Table I and Table II, the comparisons between direct mapping results and our proposed heuristics results are made under two circumstance: SOW constraint of 12 and SOW constraint of 100. While comparing with the direct mapping results, combination algorithms obviously show up their advantages. In the comparison, the smallest node counts, number of levels, and the product of node count and number of levels that can be achieved by the four proposed algorithms are selected for each benchmark. Our experiments indicate that on average, the best results that can be achieved from our proposed heuristics under SOW constraints of 12 and 100 provide a reduction of, respectively, 35.1% and 46.1% in node count, 40.6% and 49.3% in number of levels, and 58.0% and 71.2% in area delay product. When compared with results under an SOW constraint of 12, the result generated under an SOW constraint of 100 makes an improvement of 15.0%, 15.9% and 30.3% on area, delay, and area delay product, respectively.

Benchmark	Direct Mapping		Non-Stop		LP		BAB		DDAB	
	Gates	Levels	Gates	Levels	Gates	Levels	Gates	Levels	Gates	Levels
b1	13	2	9	2	9	2	9	2	9	2
cm42a	17	3	10	1	10	1	10	1	10	1
decod	27	2	16	1	16	1	16	1	16	1
cm82a	29	5	11	3	11	3	10	3	8	3
majority	7	2	7	2	7	2	7	2	7	2
parity	75	8	33	6	40	5	25	8	30	5
unreg	99	3	64	2	64	2	48	3	64	2
9symml	164	10	95	6	101	6	87	7	101	6
x2	34	3	18	2	20	2	19	2	20	2
cm152a	12	2	9	2	9	2	9	2	9	2
cm162a	60	6	27	3	28	3	20	4	26	3
cu	39	5	25	3	25	3	23	3	23	3
cm163a	50	7	19	3	20	3	17	3	20	3
cmb	36	8	16	4	16	4	17	6	16	4
pm1	54	5	28	3	28	3	21	3	28	3
tcon	33	2	16	2	16	2	16	2	16	2
pcle	58	8	29	3	31	3	27	4	31	3
pcler8	66	9	37	4	39	3	43	4	38	3
cc	63	5	25	2	25	2	25	2	27	2
i1	28	5	17	3	18	3	17	3	16	3
lal	132	7	66	4	68	4	54	5	66	4
count	143	18	74	3	74	3	57	5	74	3

TABLE I
Comparison Between Results from the Direction Mapping and the Proposed Heuristics under SOW Constraint of 12

Benchmark	Direct Mapping		Non-Stop		LP		BAB		DDAB	
	Gates	Levels	Gates	Levels	Gates	Levels	Gates	Levels	Gates	Levels
b1	13	2	9	2	9	2	9	2	9	2
cm42a	17	3	10	1	10	1	10	1	10	1
decod	27	2	16	1	16	1	16	1	16	1
cm82a	29	5	11	3	11	3	10	3	8	3
majority	7	2	7	2	7	2	7	2	7	2
parity	75	8	33	6	40	5	25	8	30	5
unreg	99	3	64	2	64	2	48	3	64	2
9symml	164	10	95	6	101	6	87	7	101	6
x2	34	3	18	2	20	2	19	2	20	2
cm152a	12	2	9	2	9	2	9	2	9	2
cm162a	60	6	27	3	28	3	20	4	26	3
cu	39	5	25	3	25	3	23	3	23	3
cm163a	50	7	19	3	20	3	17	3	20	3
cmb	36	8	16	4	16	4	17	6	16	4
pm1	54	5	28	3	28	3	21	3	28	3
tcon	33	2	16	2	16	2	16	2	16	2
pcle	58	8	29	3	31	3	27	4	31	3
pcler8	66	9	37	4	39	3	43	4	38	3
cc	63	5	25	2	25	2	25	2	27	2
i1	28	5	17	3	18	3	17	3	16	3
lal	132	7	66	4	68	4	54	5	66	4
count	143	18	74	3	74	3	57	5	74	3

TABLE II
Comparison Between Results from the Direction Mapping and the Proposed Heuristics under SOW Constraint of 100

	Previous Research		Proposed Research		
BenchMark	Node	Level	Node	Levels	Method
cm42a	13	2	10	1	Non-Stop
decod	24	2	16	1	Non-Stop
cm82a	12	3	8	3	DDAB
majority	1	1	1	1	Non-Stop
parity	45	8	30	5	DDAB
unreg	50	4	48	2	LP
cm163a	25	5	17	2	DDAB
tcon	32	2	16	2	Non-Stop
pcle	35	5	27	4	BAB
pcler8	47	6	43	4	BAB
cc	35	5	25	2	LP

TABLE III
COMPARISON WITH PREVIOUS RESEARCH UNDER FAN-IN RESTRICTION OF 6

From the comparisons, each method shows its merits and disadvantages. Basically, Bottom-up Analysis Based combination method achieves better results in node count, but it also gives larger number of levels in many cases, and in area delay product, it also does not perform well when compared with the other algorithms. Since the Dual Direction 2-pass Analysis Based (DDAB) works similarly to the Level Priority (LP) algorithms, both give good results in number of levels where the LP method is slightly better, however LP method does a little worse than the other algorithms in node count. Basically, DDAB solution performs a little better in area delay product, while the other three algorithms give similar results on this metric in most cases. Also Non-Stop method gives a stable performance in all the metrics.

A comparison between our results and the results reported in previous research [7] was also performed. As the authors of [7] utilized in their experiments a fan-in restriction of 6, a direct comparison between our results and theirs may be unfair in some cases as, under our restriction, our tool may generate gates with input number larger than 6, and under their restriction, the maximum SOW resulted by their tool might exceed our constraints. Thus we selected from our experiments only the ones satisfying their fan-in restriction of 6, and compare these results with theirs on the issues of the node count, number of levels, and area delay product.

In the comparison presented in Table III, 11 eligible benchmarks are selected, where our methods generated networks with fan-in equal or smaller than 6. Our TL synthesis tool performs better on all the three comparison metrics. According to our calculation, the average node count reduction is 24.5%, the reduction on number of levels is 32.8%, and the biggest gain is on area delay product, for which we obtained a 49.4% reduction.

VI. Conclusion

In our work, we proposed a number of TL synthesis algorithms and implemented them in a TL network synthesis tool. Our tool was successfully and seamlessly embedded into SIS. First of all, we proposed the utilization of the Sum Of the Weights (SOW) as TL synthesis constraint instead of the traditional gate fan-in. This decision made the synthesis tool better aware of the basis of TL underlying technologies. In view of the powerful capabilities of SIS in Boolean logic (BL) minimization and BL network mapping, we took the pre-processed result of BL network from SIS as input to our TL synthesis tool and made as a first step towards TL synthesis a direct mapping from BL networks to TL networks. Aware of the lack of TL network synthesis theory and the fact that the problem is NP-complete, we implemented four heuristics for the TL network optimization. The four combination algorithms make the network far more efficient than the direct mapping network. When compared with previous TL synthesis research in [7], using the same benchmarks and similar restrictions, we achieved a 25% reduction on node count, a 33% reduction on network depth, and an almost 50% reduction on area delay product on average.

References

[1] P. M. Lewis II and C. L. Coates. *Threshold Logic*. New York: Wiley, 1967.

[2] Hakan Ozdemir, Asim Kepkep, Banu Pamir, Yusuf Leblebici, and Ugur Cilingiroglu. A Capacitive Threshold-Logic Gate. *IEEE Transaction On Computer-Aided Design Of Integrated Circuits And Systems*, 31(8):1141–1149, August 1996.

[3] M. D. Padure, S. D. Cotofana, and S. Vassiliadis. Cmos implementation of generalized threshold functions. In *Proceedings of the International Work-conference on Artificial and Natural Neural Networks (IWANN2003)*, pages 65–72, June 2003.

[4] C. R. Lageweg, S. D. Cotofana, and S. Vassiliadis. A full adder implementation using set based linear threshold gates. In *Proceedings 9th IEEE International conference on electronics, circuits and systems - ICECS 2002*, pages 665–669, September 2002.

[5] K. Maezawa, H. Matsuzaki, M. Yamamoto, and T Otsuji. High-speed and low-power operation of a resonant tunneling logic gate mobile. In *IEEE Electron Device Letters*, pages 80–82, March 1998.

[6] Michael Dertouzos. *Threshold Logic: A Synthesis Approach*. M.I.T. Press, 1965.

[7] Rui Zhang, Pallav Gupta, Lin Zhong, and Niraj K. Jha. Threshold Network Synthesis and Optimization and Its Application to Nanotechnologies . *IEEE Transaction On Computer-Aided Design Of Integrated Circuits And Systems*, 24(1):107–118, January 2005.

[8] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. Sis: A system for sequential circuit synthesis. In *Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41*, May 1992.

[9] Li Zhang. Threshold logic network synthesis suite, July 2005.

[10] M. D. Padure, C. Dan, S. D. Cotofana, M. Bodea, and S. Vassiliadis. Capacitive threshold logic: A design perspective. In *22nd International Semiconductor Conference CAS 1999*, pages 81–84, October 1999.

[11] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1993.

[12] S. Muroga. *Threshold Logic and Its Application*. Wiley and Sons Inc., 1971.

[13] Krzysztof Kozminski. Benchmarks for layout synthesis - evolution and current status. In *28th ACM/IEEE Design Automation Conference*, pages 255–270, 1991.