Method of Synthesis to Identify Threshold Logic Functions

Anup Kumar Biswas
Department of Computer Science and Engineering
Kalyani Govt. Engineering College, Kalyani,
Nadia-741235, West Bengal, India

Abstract: Creation of a threshold function (TF) and identification of TF is the main thing in this work. Really, linear threshold function determines if a Boolean function is represented with the help of a threshold logic gate (LTG) or not. An efficient and effective algorithm for TF identification that we have proposed, by making the system keeping the irredundant inequalities and adjusting the minimum weight assignment. This proposed algorithm accepts the order of Chow parameters and the system of inequalities taken from a function for the purpose of assigning minimum variable weights and optimal threshold value. Our proposed algorithm becomes first heuristic algorithm not using integer linear programming (ILP) which has the capability to identify all threshold functions having 0 to 5 variables. This is the first non-ILP-based approach that can identify all the eight-input TFs. The results we have got experimentally demonstrated that our proposed process is more effective in comparison with all the present non-ILP-based approaches. The LTGs estimated by the proposed process are optimal about 100% cases. For TFs having 9–15 input variables, the proposed approach is able to identify 10^5 randomly generated TFs as well in a reasonable CPU time. As the average execution time is less than 0.001sec per function, the proposed algorithm is scalable**.** Furthermore, since the method assigns the minimum weights, as a result it provides us circuits with minimum area.

Keywords: Threshold value, TLG, weight assignments. Logic synthesis, single gate

I. INTRODUCTION

Electronic goods of Low cost, low power consumption, high operating speed, and high integration density are economically requisite in the field of business, engineering, science and technology in the present era.

MOS based transistor has a limit of scaling which inspires us to investigate the alternatives to VLSI circuit design. Some nano-devices technologies like single electron transistor (SET), resonant tunneling devices (RTD), quantum cellular automata (QCA), spintronics and tunneling phase logic (TPL) [1] are the potential candidates that can be the alternative of VLSI circuit design. These technologies show their particularities to implement the digital circuits design. In these emerging technologies, threshold logic gates (TLGs) are checked and found it suitable to such emerging technologies [2, 5, 6]. A number of implementations regarding TLGs is proposed for both new nano-metric technologies [3, 4] and CMOS.

Boolean functions synthesis using TLG is a process by which a design flow diagram is constructed on the basis of threshold logic. To perpetrate this step, lion's share methods analyze a system of conditional equations or inequalities made from the truth table of the required function, by using (ILP) [7].

This method is used to synthesize TLGs [5,6,8]. The ILP gives optimal results, but there has been the drawback of not being scalable, since whenever the number of variables enhances the number of inequalities to be solved increases exponentially.

Gowda et al., in [9], proposed one non-ILP method that can justify the condition(s) regarding threshold logic functions (TLFs) and indentify it, after that they are improved [10]. Palaniswamy et al. proposed and created a new method on the basis of modified Chow parameters, and this method is improved later in [12]. Here, the main drawbacks, we get, of these methods are the total number of identified functions as well as the assigned input weights are not always the minimum possible. As a result, final area/population of RTD circuits [6] is impacted. So the final area of the TLG based circuits [2,5] is impacted by the non-minimal weights.

One of the prior suggestions to non-ILP based methods in order to identify TLFs was suggested by Gowda et al., [9], and improved afterwards in [10]. It is really on the basis of min-max factorization tree and functional decomposition.

Our present work does the work of Synthesis and identification of TLF being performed by a non-ILP method. The proposed algorithm uses both the inequalities generated from the truth table and the Chow parameters to assign the

variable weights in the TLG. To the best of our knowledge, it is the first approach which identifies all functions with up to five variables, and is also able to identify more functions than the other non-ILP methods. It is demonstrated experimentally for up to seven variables, presenting reasonable execution time. Another feature is that the minimum input weights are assigned.

The remaining part of this paper is organized like the following. For clear understanding regarding our work, some fundamentals with respect to threshold logic and Boolean logic are presented in chapter II. The proposed method is described in detail in Chapter III. In Chapter IV, results are given; also the efficiency of algorithm is demonstrated. Conclusion about the whole work is sited in Chapter V.

II. PRELIMINARIES

To develop the concept of our proposed method, we are to introduce fundamentals of threshold logic and its properties. Unate of logic functions, irredundant sum-of-products (ISOP), cofactors are described. In addition to them, Chow parameters are explained.

A. Threshold Logic Gate (TLG)

A Boolean function, also called a logic function, can be constructed by using TLG which is a gate that can be explained in the following way.

Every input variable will have a parameter called specific weight, and the gate will have a threshold value θ. When the sum of all input variables' weights is equal to or higher than the value of θ, the TLG's output is logic 1, otherwise output is 0.This threshold gate behavior can be expressed in Equation (1) [7]:

$$G(x_i)=\begin{cases}1, & \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0, & otherwise\end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Where, w_i is the weight of each input x_i.Value of x_i will be either 0 or 1 and θ is the gate threshold value. Equation (1) can be represented by the Fig. 1.

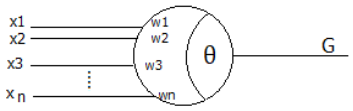

Fig.1 Threshold logic gate

B. Threshold Logic Function

Threshold logic function (TLF), of course, is a logic function whose output will be 0 or 1 and this function can be implemented with a single TLG. A TLF is a 'linearly separable' function. An n-dimensional function is said to be linearly separable when a (n-1)-dimensional space must separate all its 0-vertices from all its 1–vertices. Consider an example $f(x_1,x_2,x_3) = x_1x_3 + x_2\bar{x}_3$ which is linearly separable. A truth Table for this function is shown in Table-1. With the help of this table, if we draw a three dimensional figure for the points (0,0,0), (0,0,1),…,(1,1,1) and indicate all the points by a small colors/bubbles, when the values of this function are 1 the bubbles are pointing by green color otherwise the bubbles are colorless. All the three dimensional green points can be separated from the colorless bubbles by a two dimensional space as shown in Fig.2. If we construct a compact vector $\{x_1,x_2,\ldots,x_n : \theta\}$ by using the function's input variables and threshold value θ, we can implement TLF by this vector. For instance, the function $f(x_1,x_2,x_3)$= $(x_1 \wedge x_2 \wedge x_3)$ has a compact vector = {1,1,1:3} and the function $f(x_1,x_2,x_3)$= $(x_1 \vee x_2 \vee x_3)$ has a compact vector = {1,1,1:1}.

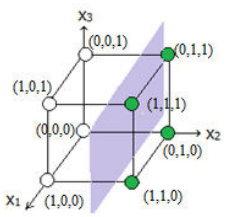

Fig. 2 Space diagram with a hyper plane of linear threshold logic function

Table-1
Truth table of $x_1x_3+x_2\bar{x}_3$

x_1	x_2	x_3	f(x_1, x_2,x_3)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

For a more complex function $f(x_1,x_2,x_3,x_4,x_5,x_6,x_7) = x_3x_6 + x_2x_5x_7 + x_1x_2x_3x_4 + x_1x_2x_4x_6 + x_3x_4x_6x_7 + x_2x_4x_5x_6$, we will be able to represent it by a LTG whose compact vector is {2, 4, 8, 5, 6, 7, 3 : 15}. For this reason, an important advantage of threshold logic is to reduce the number of gates used in the circuit, decreasing area [6].

C. Cofactors

Consider a function $f(x_1, x_2,..,x_n)$ which has input variables $(x_1, x_2,\dots, x_n)$, the cofactor f_{xi} is expressed by the equation (2) as:

$f_{x_i} = \{f(x_1, x_2\dots,x_n) \mid x_i=p, p\in B, B = (0, 1)\}$ ..... (2)

Positive cofactor with respect to x_i is defined when $k = 1$, and negative cofactor is defined with respect to x_i when $k = 0$. *i.e.;*

Positive Cofactor with respect to x_1: $f_{x_1} = f(\mathbf{1}, x_2,..,x_n)$

Negative Cofactor with respect to x_1: $f_{x_1} = f(\mathbf{0}, x_2,..,x_n)$

D. Unateness

If every input variables of a function is either always positive or always negative, then the function is defined as a unate function.

For example, an OR function of two variables x and y i.e f(x, y) = x ∨ y is unate (since x and y are positive unate)

The function f(x, y) = x⊕y (XOR) is not unate because it changes its behavior depending on the values of both x and y, not just one of them.

In the context of Boolean algebra and logic, an unate variable is a variable that appears in a sum-of-products (SOP) expression either only in its positive form (e.g., x) or only in its negative form (e.g., '$\overline{x}$'), but not both. If it appears in both forms, it's considered binate or mixed. A function is considered unate if it's unate in all of its variables.

When all of the variables of a function is unate then the function is called unate. Let $U(f, x_k)$ denote the unateness detection function of an input variable x_k at function f, and auxiliary function

$i = f_{x_k=1} \vee f_{x_k=0}$, one has:

$$U(f,x_k)=\begin{cases}\text{positive unate} & (f_{x_k=1}\equiv i)\wedge(f_{x_k=1}\neq f_{x_k=0}) \quad (3)\\ \text{negative unate} & (f_{x_k=0}\equiv i)\wedge(f_{x_k=1}\neq f_{x_k=0}) \quad (4)\\ \text{don't care} & f_{x_k=1}\equiv f_{x_k=0} \quad (5)\\ \text{binate} & (f_{x_k=1}\neq f_{x_k=0})\wedge(f_{x_k=1}\neq i)\wedge(f_{x_k=0}\neq i) \quad (6)\end{cases}$$

E. Logic Functions Representation

An expression is called sum-of-products (SOP) when such expression corresponds to product terms joined by a sum operation. An irredundant sum-of-products (ISOP) is a SOP where no product term can be deleted or simplified without changing the logic function. An Unate function will present only a single ISOP [13].

F. Logic Function Classes

Given a set of all functions with up to n variables, they can be grouped in classes of functions. We will be able to group the Boolean functions by the permutation of inputs (X_{per}), negation inputs (X_{neg}), and/or negation of output (Y_{neg}).

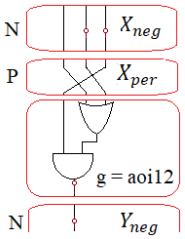

Fig.3 Types of Boolean equivalence used to group functions in classes [14].

G. Threshold Logic Functions Properties

All TLFs are unate functions, but the reverse is not true, that is not all unate functions are TLF [7]. Therefore, if a function has binate variables, the function cannot be TLF. A simple example of an unate function is $f = (x_1x_2) \vee (x_3x_4)$, it is not TLF.

For a logic function bears only complemented variables, we can manipulate the variables in the same manner as the function containing only non-complemented variable for the purpose of identification if the function is a TLF or not. If $f(x_1,x_2,\ldots,x_n)$ is TLF, defined by $\{w_1,w_2,\ldots, ,w_n: T\}$, then its complement $f'(x_1,x_2,..,x_n)$ is also TLF, defined by $[-w_1,-w_2,\ldots,-w_n: (1\text{-}T)]$.

If a NOT gate or inverter is in our hand and a function of input variables x_i [i=1,2…,n] , where inverter is TLF, then selectively complementing the inputs variables x_i we will be able to obtain a realization only using TLG that has only positive weights. This propriety is illustrated in Fig. 3 [7].

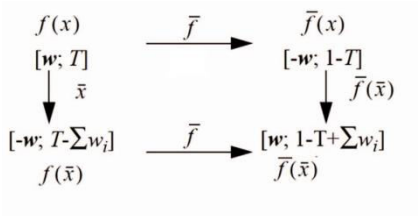

Fig. 3 Elementary properties of threshold logic [7]

H. Chow Paramaters

Chow parameters represent a particular set of parameters used to define the relationship among the weights of TLF. Given a function $f(x_1,x_2,\ldots,x_n)$, we call m_i the number of entries and for which the value of $x_i = 1$ we have $f(x_i) = 1$; and n_i the number of entries for which $f(x_i) = 1$ when $x_i = 0$. The Chow parameter p_i of a variable x_i is defined by [7]:

$$p_i = 2(m_i - n_i) \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

For instance, Fig. 4 shows the Chow parameter computation, considering the function $f = x_1 \wedge (x_2 \vee x_3)$.

The **correlation** between two parameter values p_i and p_j $[p_i \neq p_j]$ for the two input variables (x_1, x_2) induces the correlation of the weights w_i and w_j of the input variables x_i and x_j, respectively, if $p_i > p_j$ then $w_i > w_j$ [7].

x_1	x_2	x_3	f	*Value of* p_i
0	0	0	**0**	$m_1 = 3$ and $n_1 = 0$
0	0	1	**0**	$p_1 = 2(m_1 - n_1) = 6$
0	1	0	**0**	
0	1	1	**0**	$m_2 = 2$ and $n_2 = 1$
1	0	0	**0**	$p_2 = 2$
1	0	1	**1**	
1	1	0	**1**	$m_3 = 2$ and $n_3 = 1$
1	1	1	**1**	$p_1 = 2$

Fig. 4 Calculation of Chow parameters value for *f*.

Parameter Modified by Chow

The input(s) of a TF (Threshold function) correspond(s) to a set of weight(s) in its LTG representation. Hence, there exists a weight ordering among all inputs of a TF. Muroga [17] proposed the *modified Chow's parameter*, which is used to determine the variable ordering of a TF 'F'. The formula of modified Chow's parameter is listed in the following equation:

$$q_i(F) = 2 \times p_i(F) - p_0(F) \ldots\ldots\ldots\ldots\ldots\ldots (8)$$

where $q_i(F)$ represents the parameter of the i^{th} input variable of TF 'F' , $p_i(F)$ is the number of minterms in the on-set for which $x_i = 1$, and $p_0(F)$ is the number of minterms in the on-set of this function (i.e. how many 1s in the truth table of 'F'). By sorting the parameters $q_i(F)$ in the descending order for i= 1,2,…, n, we can obtain the corresponding variable ordering of an n-input TF.

For example, given $F=x_3 + x_1x_2$, its truth table is shownin Fig. 5. The number of minterms for which F =1, $p_0(F) = 5$. Three of them are with $x_1= 1$, three of them are with $x_2= 1$ and four of them are with $x_3= 1$. So, all the values of three terms $p_1(F)$, $p_2(F)$, and $p_3(F)$ are 3, 3, and 4, respectively. By (8), we can obtain $q_1(F) =[2 \times p_1(F)-p_0(F)]=[2\times3-5]=1$, $q_2(F) = 1$, and $q_3(F) = 1$. So, we can write ordering of the variables as: $w_1=w_2< w_3$.

x_1	x_2	x_3	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Fig. 5 Table of $(x_1x_2+ x_3)$

III. PROPOSED METHOD

The common strategy we are using to identify TLF is to create a system having inequalities from the truth table, and then we must solve this system by using ILP. If there is a possible solution, the function is TLF, and the solutions of the system correspond to the input weights. If the system of inequalities cannot be solved, the function is not TLF, and it requires more than one TLG to be implemented. Threshold networks will be generated on future work.

The complete system of inequalities, in the proposed method, is also created using similar course to ILP inequalities creation algorithms. However, unlike the ILP approach, the inequalities system is not solved. The proposed algorithm selects some of the inequalities as constraints to compute the input weights. In the next, after assigning the weights, the consistency of the complete system is verified, in order to check if the weights have been correctly assigned.

As mentioned before in the section II, if a function is not unate, then this function is not TLF. Therefore, first of all the algorithm performs a test to check whether the function is unate. A -ve unate variable can be changed into a positive variable if the weight signal is just inverted, and this amount is subtracted from the threshold value. Without any change of generality, we can consider whether the proposed algorithm is started from a positive unate function.

For the purpose of providing a clear understanding, the algorithm is divided into seven steps, that are given in the following steps: (1) it gives the Chow parameters computation and defines the ordering of the variable weights; in step (2), it demonstrates how to generate the inequalities system, in step (3), how to simplify these inequalities; in step (4), it is explained about the dependence map created between the variables and inequalities; in step (5), the input weight assignments are presented, in step (6), it shows how the verification that ensures the correct result is performed; finally, in step (7), the calculation of the threshold value is done. The proposed flow comprising these stages is shown in Fig. 6.

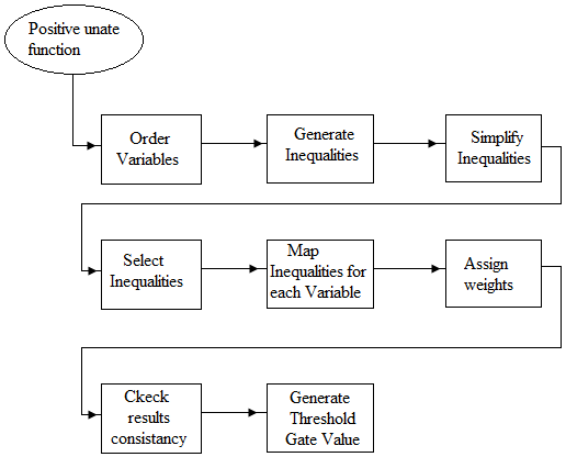

Fig.6 Flow diagram of proposed method Algorithm

Already we have informed above the ordering of Chow parameters which inherently corresponds to the ordering of the weights of the variables. Ordering of the weights is taken in the ascending order.

Whenever the Chow parameter values of two variables will be of the same values, then weights of the variables will be also the same. Accordingly, the Chow parameter values are segmented as their Chow parameter values.

For clear understanding, a figure is depicted in Fig. 4, It is shown that the Chow parameters of three variables x_1, x_2 and x_3 are calculated next to Fig.4 and the values are 6, 2 and 2, respectively. Here, firstly, weight of the 2nd and 3rdvariables x_2 and x_3 *are assigned according to our algorithm. Then* the weight of the variable $x_1$is assigned.

B. Step 2: Inequalities System Generation

From the equation $F(x_1, x_2, x_3) = x_1(x_2 + x_3)$, it is possible to create 8 conditional equations that include weight variables as well as threshold value. With an example the process is explained.

Given the function $F(x_1, x_2, x_3) = x_1(x_2 + x_3)$, the truth table of this function is shown in Table-6 where the relationship between weights and threshold value is shown also.

With the help of the Table-6, the truth Table-7 will be created according to the greater value side set and lower value side set with respect to the threshold value θ.

In Table-7, two sets of sum of weights values are created. When the sum of weights is greater than or equal to threshold θ, we keep these sum in column 1. And in column 3, when the sum of weights is lesser than threshold θ, they are kept.

In column 2 in Table-8 all the 3 summation parts [(w_1+w_3), (w_1+w_2) and $(w_1+w_2+w_3)$] are greater than threshold θ; and the threshold θ is always greater than the 4th column's value. So, we can rewrite the table-7 in an extended form like Table-8.

Table 6
Truth table of $x_1(x_2 + x_3)$, and input weights and threshold value relation.

Sl.	x_1	x_2	x_3	f	θ
1	0	0	0	**0**	$0 < \theta$
2	0	0	1	**0**	$w_3 < \theta$
3	0	1	0	**0**	$w_2 < \theta$
4	0	1	1	**0**	$w_2+w_3 < \theta$
5	1	0	0	**0**	$w_1 < \theta$
6	1	0	1	**1**	$w_1+w_3 \geq \theta$
7	1	1	0	**1**	$w_1+w_2 \geq \theta$
8	1	1	1	**1**	$w_1+w_2+w_3 \geq \theta$

Table-7 greater and smaller sets of inequalities

	Threshold θ	lower value side Set
w_1+w_3	> θ >	0
w_1+w_2	> θ >	w_3
$w_1+w_2+w_3$	> θ >	w_2
	θ >	w_2+w_3
	θ >	w_1

+Table-8

Sl.	*greater set*		*lesser set*
1	w_1+w_3	>	0
2	w_1+w_3	>	w_3
3	w_1+w_3	>	w_2
4	$\mathbf{w_1+w_3}$	>	$\mathbf{w_2+w_3}$
5	$\mathbf{w_1+w_3}$	>	$\mathbf{w_1}$
6	w_1+w_2	>	0
7	w_1+w_2	>	w_3
8	w_1+w_2	>	w_2
9	$\mathbf{w_1+w_2}$	>	$\mathbf{w_2+w_3}$
10	$\mathbf{w_1+w_2}$	>	$\mathbf{w_1}$
11	$w_1+w_2+w_3$	>	0
12	$w_1+w_2+w_3$	>	w_3
13	$w_1+w_2+w_3$	>	w_2
14	$w_1+w_2+w_3$	>	w_2+w_3
15	$w_1+w_2+w_3$	>	w_1

C. Step 3: Inequalities Simplification

All the relationships shown in Table 7 are not useful. Consider the two relation $[(w_1+w_2)\geq \theta]$ and $[(w_1+w_2+w_3)\geq \theta]$, in these relations the input weights are always positive. But the relation $[(w_1+w_2+w_3)\geq \theta]$ has one extra element w_3 which is redundant when we compare it with the relation $[(w_1+w_2)\geq \theta]$. Such type of redundancies are avoided in the proposed method using the ISOP expression of the function. Considering the example expressed in Table-6, the algorithm generates the inequalities 4, 5, 9 and 10 only from Table-8 after avoiding the redundancies.

By generating only the irredundant inequalities like 4, 5, 9 and 10 in Table-8, the algorithm will be able to simplify each of them when necessary. This simplification happens when an element, here weight, is found on both sides of the same inequality expression. For instance, say the inequality 4 in Table-8, i.e.,$[w_1+w_3]> [w_2+w_3]$, the algorithm simply it and provide the inequality $[w_1>w_2]$ after removing w_3. As all the weights are positive, our algorithm will remove such inequalities bearing no weight (or '0') in the *lesser side*.

D. Step 4: Inequalities Selection

For selecting the input variable weights, we must select those inequalities that have simply only a single weight within the greater side. This weight should be greater than all variable weights within the right side /smaller side. Such technique is necessary for the step called 'weights assignment' discussed in the next step. The selected inequalities 4 and 9 in Table-8 are chosen and after simplification we have only $[w_1>w_2]$ and $[w_1>w_3]$.

From the simplified inequalities, the left-hand input weight is said to be the ***key*** of the inequality. The proposed algorithm delivers a relation among inequalities having the same ***key***. In this situation, the *keys* for the two inequalities are the same weight w_1 , where the inequalities are $[w_1>w_2]$ and $[w_1>w_3]$.

E. Step 5: Weights Assignment

We will assign the input weight w_i only on the basis of inequalities selected in the above discussion where all variables are related. The proposed algorithm willingly generates a temporary variable key which must manage the weights assigned to.

At the initial point, this temporary key is assumed to be a minimum integer = 1 and this value is increased by +1 at each iteration until a good solution is obtained.

At every instant the algorithm assigns weight for the variables and the inequalities concerned are checked. If the current assigned value for the ***key*** saturates the inequalities involved, this value is treated as the weight of the variable. If does not satisfies, the value is increased by (+1) and the inequalities are checked. This process is repeatedly done until a satisfactory limit is obtained. The same action is performed for each variable.

Flow diagram of weights assignment process is shown in Fig. 7. For the example cited in Table-6, the assignment is fixed on the two weights w_2 and w_3. As no constraints are found for these keys on inequalities, the value '1' is assigned or fixed. Now, we increase the temporary variable to 2 and assign it to w_1. For this temporary variable or key 2 we have two inequalities as $[w_1> 1]$ and $[w_1> 1]$ for this key value=2 and we put $w_1 = 2$ and we obtain two equal true inequalities $[2 > 1]$ and $[2 > 1]$. As all the weights are assigned for obtaining true inequalities, this step ends in. The weights selected and assigned for the weights in the function $F(x_1, x_2, x_3) = x_1(x_2 + x_3)$ are w_2=1 , w_3=1 and w_1=2.

The algorithm used here always creates minimum possible input weights and they are then assigned. As a result, the TLG implementation is synthesized with the help of minimum circuit components. The advantage underlying here can be shown using the example. For a given function $f = x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_3+x_1x_5x_6$ the Palaniswamy's algorithm [12] informs us it to be a TLF and assigns the weight $[w_1w_2w_3w_4w_5w_6: \theta]$=[9,8,4,4,1,1: 11], But when using the proposed algorithm the weights are assigned $[w_1w_2w_3w_4w_5w_6: \theta]$=[7,6,3,3,1,1; 9] and it is the minimum possible set[16].

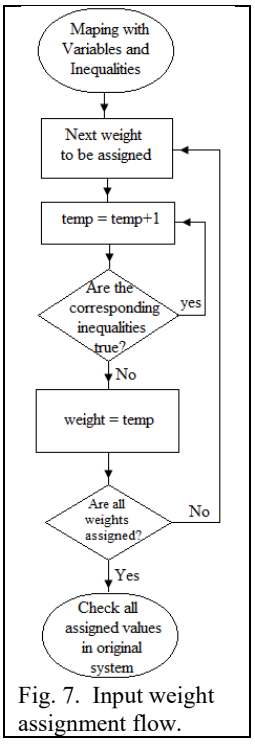

Fig. 7. Input weight assignment flow.

IV. EXPERIMENTAL RESULTS

For the purpose of validating and comparing the proposed algorithm to the existing ones available, 2 experiments are performed. The first one calculates the efficiency present in the proposed process considering total number of TLF which are identified. The second experiment calculates the processing time by evaluating the mean time considering for all identified TLF. The proposed algorithm is developed using Java in this case.

Total number of TLF identified and to be maximized is the target of TLF-identification algorithms. Under the consideration of total number of TLF identified, the first experiment determines the efficiency. The count of identified TLF for the variable numbers starting from 2 to 8 has already been established by Muroga [16]. As reference these evaluated numbers are accepted.

Instead of using linear programming to identify TLF, Gowda et al.[9] have accepted the first method and again improved it next[10]. Palaniswamy *et al.*[12] presented an approach that gives better results. After justification, result of Palaniswamy's approach has been considered as the basis of reference and comparison when we are developing a method.

Experiment results are cited in Table-9 for three cases: Moruga[16], Palaniswamy [12] and Our approach. In the second column in Table-9, integer numbers indicates how many TLFs are created for Moruga [16], against how many numbers of inputs are there. In column 2, shows the quality of function identified by Palaniswamy [12]. In column 3 it is shown the number of TLFs identified in the case of our proposed method.

We have observed that for input variable number 1 to 4 all the methods identify each TLF existing. When the variable number =5, Palaniswamy's process identifies 84 existing TLFs out of 92, but our proposed method identifies all existing TLFs. When six variables are considered, Palaniswamy's process identifies 73.2% of TLF and the proposed method does 90.4% of TLF. When seven variables are taken into consideration, the identified TLFs are 32.6% and 59.4% for the cases of Palaniswamy's process and our proposed process respectively. The proposed algorithm can be considered as pioneer non-ILP algorithm and this can identify all TLFs accurately for all inputs upto 5 (from 1through 5) and provides the superior results with respect to 'state-of-the-art algorithm'.

Table-9 Proposed method efficiency

Number of NP classes identified TLF by each method					
Input number	Moruga[16]	Palaniswamy [12]		Our approach	
	TLF	TLF	%	TLF	%
1	1	1	100	1	100
2	2	2	100	2	100
3	5	5	100	5	100
4	17	17	100	17	100
5	92	84	91.3	92	100
6	994	728	73.2	900	90.4
7	28262	9921	32.6	16804	59.4

The execution time is evaluated, in the second experiment, by calculating the mean time consumed to identify every TLF. It is clear that our proposed algorithm determines all the numbers of TLFs for all variables from 1 to 7. These numbers inform that the proposed algorithm is so fast as Palaniswamy's method is. From the Table-10, it is seen that the average execution time does not increase exponentially whereas ILP-based algorithms does.

Table 10 Execution time per identified TLF

Average time per function for each number of inputs							
Inputs	1	2	3	4	5	6	7
Time Reqd. (ms)	0.3	0.2	0.3	0.4	0.5	0.4	0.6

With the increasing of the invariables, solving the conditional equations by linear programming will be more complex and expensive. From the Fig. 8 we have observe "the scalability for consumed average time per function of the proposed algorithm" and "the exponentially behavior of the integer linear programming (ILP) algorithm". The mean execution

time for each function is measured for all number of variables. The variable number starts from 1 to 16 and the results acquired are shown in a logarithm scale.

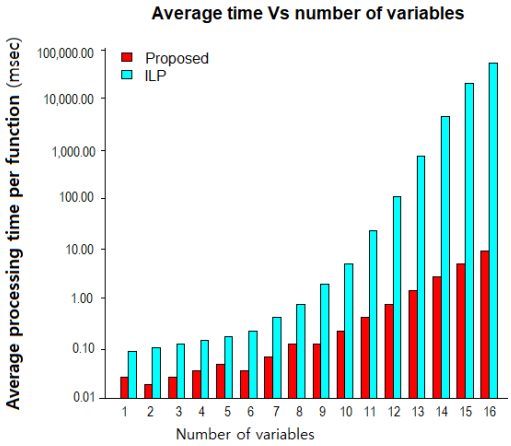

Fig. 8 – Average execution time per function for each number of variables

The graph of execution time for ILP exponentially increased when the input variables goes from 8 towards 16 and reaches tens of seconds, whereas the proposed algorithm goes only a few milliseconds for the same number of variables.

The ILP method execution time has a marked increases from eight variables, and reaches tens of seconds for sixteen variables. In the other hand, the proposed algorithm reaches only some milliseconds, for this number of variables.

When we comparing the two curves in Fig. 8, we observe the time difference is close to 4 magnitude orders. Yet our proposed approach does identify accurately all TLF about the present benchmark and gives enhanced algorithm efficiency.

V. CONCLUSION

Based on the threshold logic function, threshold logic gate is constructed which helps us to implement a nano-level circuit requiring a minimum number of components. As a result, the population density of complex circuits will be higher when comparing with the MOS based circuits. Here a threshold logic function is proposed by a novel algorithm called “non-ILP algorithm”. The proposed algorithm uses inequalities generated from the truth table, but does not use linear programming. The input weight assignment uses some of the inequalities as constrains and it is performed in ascending order, defined by Chow parameters. These weights assigned are always the minimum, since uses a bottom-up approach to compute the weights, impacting the final area of circuits. From the results we can conclude that our method can strongly identify all the variables starting from 1 through 5. Although being heuristic, our algorithm identifies more functions than the other heuristic methods when the number of variables increases. Average execution time per function for each number of variables is scalable in our case, but not for the case of ILP. Whenever any TLF in implemented by using TLG based on an emerging technology called single electron threshold gate, the speed of the circuit will be enhanced at least 4 times with respect to classical CMOS circuit.

VI REFERENCES

[1] “Semiconductor Industries Association Roadmap.” http://public.itrs.net 2011.
[2] Zheng, Y. Novel RTD-based Threshold Logic Design and Verification. Master’s thesis, Virginia Polytechnic Institute and State University, 2008.
[3] Celinski, P. et al., 2003, “State of the art in CMOS threshold logic VLSI gate implementations and systems”, in Proc. of the SPIE, pp. 53–64.
[4] Beiu, V., Quintana, J. M. and Avedillo, M. J., Sep. 2003, “VLSI implementations of threshold logic: A comprehensive survey,” IEEE Trans. Neural Netw., vol. 14, no. 5, pp. 1217–1243.
[5] Avedillo, M. J. and Quintana J.M., 2004, “A threshold logic synthesis tool for RTD circuits,” in Proc. Euromicro Symp. Dig. Syst. Des., pp. 624–627.

[6] Zhang, R., Gupta, P., Zhong L. and Jha, N.K., 2005, “Threshold network synthesis and optimization and its application to nanotechnologies,” IEEE Transactions on CAD, vol. 24, no. 1, pp. 107–118.
[7] Muroga S., 1971, “Threshold Logic and Its Applications” New York: Wiley-Inter science.
[8] Subirats, J., Jerez, J. and Franco, L., Nov. 2008, “A new decomposition algorithm for threshold synthesis and generalization of boolean functions”, IEEE Trans. Circ. Syst. vol. 55, no. 10, pp. 3188–3196.
[9] Gowda, T., Vrudhula, S. and Konjevod, G., 2007, “A non-ilp based threshold logic synthesis methodology”, In Proc. IWLS.
[10] Gowda, T. and Vrudhula S., 2008, “A decomposition based approach for synthesis of multi-level threshold logic circuits”, Proc. of ASP-DAC, pp. 125-130.
[11] Palaniswamy, A.K., Goparaju, M.K. and Tragoudas, S. 2010 “Scalable identification of threshold logic functions”, in Proc. GLSVLSI. 269–274.
[12] Palaniswamy, A.K., Goparaju, M. K., and Tragoudas S., Aug 2012, “An Effcient Heuristic to Identify Threshold Logic Functions”, In ACM Journal on Emerging Technologies in Computing Systems, Vol. 8, No. 3,pp. 19:1–19:17.
[13] Brayton, R.K., McMullen C. , Hachtel G. D. and Vincentelli, A S., “Logic Minimization Algorithms for VLSI Synthesis”, 1984:Kluwer Academic Publishers.
[14] Hinsberger, U. and R. Kolla 1998, "Boolean matching for large libraries", in Proc. Design Automation Conf., pp.206 -211
[15] Avedillo M. J., Quintana J.M., and Rueda, A.: "Threshold Logic" In the "Encyclopedia of Electrical and Electronics Engineering", John Webster(Ed.) 1999, John Wiley & Sons. vol. 22, pp. 178-190.
[16] Muroga S., Tsuboi T. and Baugh, C. R., 1970, “Enumeration of Threshold Functions of Eight Variables,” IEEE Trans. Comput vol. C, no. 9, pp. 818–825.
[17] S. Muroga, *Threshold Logic and Its Applications*. New York, NY, USA: Wiley, 1971.