An Efficient Method to Threshold Logic Functions Identification

Augusto Neutzling¹; Mayler G. A. Martins²; Renato P. Ribas¹²; André I. Reis¹²

¹PPGC /²PGMICRO, Institute of Informatics, UFRGS, Porto Alegre, RS, Brazil.
(ansilva, mgamartins, rpribas, andreis}@inf.ufrgs.br

Abstract - In this paper, a novel method to identify threshold logic functions (TLF) is proposed. Threshold logic is an alternative to conventional Boolean logic, and it is being revisited due to its suitability in emerging technologies, such as QCA, RTD, SET, TPL and Spintronics. Identification and synthesis of TLF are fundamental steps for the development of circuit design flow based on threshold logic. The proposed method exploits both the order of Chow parameters and the system of inequalities to assign the variable weights and the threshold value. This is the first algorithm that not uses integer linear programming (ILP) and is still able to identify all threshold functions up to five variables. Moreover, it also identifies more functions than other methods when the number of variables is higher than five. The average execution time is less than 1 ms per function, similarly to the state-of-the-art approaches, and the proposed algorithm is scalable. Furthermore, the method always assigns the minimum weights, which results in circuits with less area overhead.

Keywords: Threshold logic, RTD, QCA, Spintronics.

I. INTRODUCTION

The limits of MOS transistor scaling make necessary the investigation of new alternatives. Among the main proposals are the nano-devices such as quantum cellular automata (QCA), resonant tunneling devices (RTD), single electron transistor (SET), tunneling phase logic (TPL) and Spintronics [1]. Some of these new technologies have particularities in the design of circuits due to the most appropriate design style to implement logic gates. They are more suitable to build threshold logic gates (TLG) than the basic CMOS gates [2].

Several implementations of TLGs have been proposed for both CMOS technology, and new nanometric technologies [2][3]. One of the most mature technologies in this scenario is the RTD [4-6]. Fig. 1 shows a threshold logic circuit using RTDs, where the value of the weights is proportional to the area of devices.

In order to achieve a complete design flow based on threshold logic, a crucial step is the synthesis of Boolean functions using TLGs. This step defines if the function can be implemented in a single TLG and assign weights to each input. To accomplish this step, most of the methods solve a system of inequalities generated from the truth table of the function, using integer linear programming (ILP). Such solution is discussed by Muroga, in [7], being used to synthesize threshold logic by Avedillo and Quintana [4], by Zhang et al. [5] and by Subirats et al. [8]. However, such approaches have the bottleneck of not being scalable, because when the number of variables increases the number of inequalities to be solved increases exponentially.

The first known non-ILP based method to identify threshold logic functions (TLF) was proposed by Gowda et al., in [9], and improved in [10]. It is based on functional decomposition and min-max factorization tree. In [11], Palaniswamy et al. proposed a new method, which is directly based on the modified Chow parameters. However, the drawback of these methods is the number of identified functions and that the assigned weights are not always the minimum possible, so impacting the final area of RTD circuits [6].

This paper proposes a new method to perform the identification and synthesis of TLFs without using linear programming. The proposed algorithm uses both the inequalities generated from the truth table and the Chow parameters to assign the variable weights. To the best of our knowledge, it is the first approach which identifies all functions up to five variables and identifies more functions than the other non-ILP methods, up to seven variables, at similar execution time. Another advantage is that the minimum weights are encountered, which results directly in smaller area.

The rest of this paper is organized as follows. In Section II, we present some fundamentals about threshold logic and Boolean logic for a good understanding of the work. The proposed method is described in detail in Section III. Section IV presents the results, showing the algorithm efficiency. Finally, the conclusions are discussed in Section V.

Figure 1 - RTD threshold logic gate [6].
II. PRELIMINARIES

Some concepts are essential for a good understanding of the proposed method. In this section, the fundamentals of threshold logic and some of its properties are presented. Moreover, Chow parameters are explained, as well as basic concepts of Boolean functions such unateness.

A. Threshold Logic Gate

A **threshold logic gate** (TLG) is a gate that implements a Boolean function using a different way from the traditional one, as follows. Each input has a determined weight, and the gate has a threshold value. If the sum of the weights of the ON inputs that is equal to or greater than the threshold value the output gate is ‘1’. Otherwise, the output is ‘0’. This operating principle can be given by [7]:

\[
  f = \begin{cases} 
  1, & \sum_{i=1}^{n} w_i x_i \geq T \\ 
  0, & \text{otherwise} 
  \end{cases}
\]  

(1)

where \(x_i\) represents each input, \(w_i\) is the weight of each input, and \(T\) is the threshold value.

B. Threshold Logic Function

A **threshold logic function** (TLF) is a Boolean function that can be implemented by using a single TLG. The TLF can also be called linearly separable. A TLF is completely represented by a compact vector \([w_1, w_2, \ldots, w_n; T]\). TLF can be complex a function, such as \(f = x_1 x_2 \lor x_1 x_3 \lor x_2 x_3 \lor \overline{x_4} \lor x_5\) which corresponds to the TLG \(f = [4,3,3,1,1;7]\). For this reason, an important advantage of threshold logic is to reduce the number of gates used in the circuit, decreasing area [6].

C. Unateness

An unate function is a Boolean function represented by a formula such that each variable appears either in the positive or in the negative form throughout the formula. Read-once functions are unate, but \(f = x_1 x_2 \lor x_1 x_3 \lor x_2 x_3\) is an example of a unate function which is not a read-once function.

D. Threshold Logic Properties

All TLF are unate functions, but not all unate functions are TLF [7]. Therefore, if a function has binate variables, the function cannot be TLF. The function \(f = x_1 x_2 \lor x_3\) is a simple example of an unate function that is not TLF.

If the given logic function contains complemented variables, these variables can be manipulated exactly as the same way as functions having all non-complemented variables, to identify if it is a TLF. If \(f(x_1, x_2, \ldots, x_n)\) is a threshold function defined by \([w_1, w_2, \ldots, w_n; T]\) then its complement \(f(x_1, x_2, \ldots, x_n)\) is also a threshold function defined by \([-w_1, -w_2, \ldots, -w_n; 1-T]\). If a function is a TLF, then selectively complementing the inputs, it is possible to obtain a realization by a TLG with only positive weights [7].

E. Chow Parameters

The **Chow parameters** comprise a particular set of parameters used to define the relation among the weights of a TLF [7]. The variable with a smaller chow parameter has a smaller weight.

Given a function \(f(x_1, x_2, \ldots, x_n)\), the Chow parameter of a variable \(x_i\) is defined by the twice the difference of the number of entries for which \(x_i = 1\) and \(f(x_i) = 1\), and the number of entries for which \(x_i = 0\) and \(f(x_i) = 1\) [7].

III. PROPOSED METHOD

As showed in Subsection II-D, if a function is not unate, then this function is not a TLF. Therefore, the algorithm first performs a test to check whether the function is unate. A negative variable can be changed into a positive variable if the weight signal is inverted and this amount is subtracted from the threshold value. From these statements, it is possible to consider the algorithm in this paper receiving only positive unate functions, without loss of generality.

The proposed method consists in an assignment of weights based on the variables weights ordering, and the inequalities generated from a truth table. The weights are assigned to each variable in ascending order, and after each assignment the consistence of the respective inequalities is verified. If the inequalities are not satisfied, the weight assigned is increased. A map is created by associating each variable with inequalities which has dependencies.

In order to provide a simpler understanding, the algorithm is split in five stages, which are summarized in the following subsections. The first and second steps start from a positive unate function in the ISOP format. Subsection A shows the definition of the ordering of the variable weights, based in Chow parameters computation, and Subsection B explains the inequalities system generation and simplification. Subsection C presents the creation of a dependence map between the variables and inequalities, that are used as constrains in the next step. The weights assignment is explained in Subsection D and in Subsection E is shown how works the verification that ensures the correct result, and how the threshold value is obtained. The proposed algorithm flow containing each stage is shown in Fig.2.

![Diagram](image-url)
A. Ordering Variable Weights

As shown in Section II, the ordering of the Chow parameters corresponds to the ordering of the variable weights. This is used to define the ascending order which the weights should be assigned.

The proposed algorithm assumes that the Chow parameter value of two variables are the same, the variables must have equal weight. Based on this statement, the variables are grouped by their Chow parameter value.

Given the function \( f = x_1 (x_2 + x_3) \), the calculated Chow parameters of the variables are \( x_1, x_2 \) and \( x_3 \) are 6, 2 and 2, respectively. In this case the algorithm assigns first the weight of the variables \( x_1 \) and \( x_3 \) and after the weight of the variable \( x_2 \).

B. Inequalities Generation

The most common method used to identify TLFs is generating a system of inequalities from the truth table and solving this system using integer linear programming (ILP). If there is a possible solution, the function is TLF and the solutions of the system correspond to the weights. If the system of inequalities cannot be solved, the function is not TLF. In the proposed method, the inequalities are generated, and some of them are used as constraints to weights assignment. After that, the consistency of all inequalities is verified.

The generation of inequalities is based on the principle of threshold logic. According to Equation (1), the sum of weights of each onset cube needs to be greater than the threshold value, and the sum of weights of each offset cube needs to be smaller than the threshold value. Therefore, the sum of weights of each onset cube has to be greater than the sum of weights of each offset cube \([7]\). The algorithm creates each inequality, combining each onset cube with each offset cube.

The method uses ISOP, to avoid the generation of redundant inequalities. Besides creating the irredundant inequalities, the algorithm simplifies each of them when possible. The simplification occurs when the same variable (or two different variables that have the same weight), appearing on both sides of some inequality. When this happens, this variable is removed.

C. Map Inequalities

Only the inequalities that have just one variable of the greater side are used in the assignment. This ensures that in the assignment step the smaller sides of inequality have already assigned weight to all variables. That is because all variable weights of smaller side are smaller than the weight variable from the greater side. This variable weight is the key of that inequality. Each key can point to several inequalities.

D. Weights Assignment

The weight assignment is done using only the selected inequalities that are related to each variable. The algorithm creates a temporary variable, which controls the value assigned to the weights. This variable is initialized with a minimum value (in the first time is `1`) and is increased by one during the iterations. The order of assignment is the same as the one defined by Chow parameters, starting with the variables that have the lowest weight and continuing in ascending order, ensuring that the minimum weights are assigned.

Each time a weight is assigned, the consistency of the corresponding inequalities are checked. If the current value of the temporary variable satisfies the inequalities, this value is indeed the weight of the variable. Otherwise, the value is increased to satisfy the inequalities or until an upper limit defined proportionally by the number of variables is reached.

This procedure is repeated for each variable. Fig. 3 shows the flow for weights assignment procedure.

This bottom-up approach used to compute the input weights, ensure the weights assigned by the algorithm are always the minimum possible, allowing the TLG implementation be synthesized using less area. This advantage can be observed in the following example. Given the function \( f = x_1x_2 \lor x_1x_3 \lor x_1x_4 \lor x_2x_3 \lor x_2x_4 \lor x_1x_5x_6 \), the Palaniswamy’s algorithm \([11]\) identifies the TLF, assigning the weights \([9,8,4,4,1,1;11]\), whereas using the proposed method assigns \([7,6,3,3,1,1;9]\), which is the minimum possible \([12]\).

E. Checking Weights on Original System

The algorithm performs a check to ensure that the values encountered are a valid solution of the system. All the inequalities of the original system are checked with the weights that were defined by the algorithm for each variable.

If all inequalities are consistent, the function is defined as a threshold logic function, and the weights assigned are accepted. If at least one of the inequalities is false the function is defined as not being TLF. This ensures that the algorithm does not result in a false positive TLF. After assign the weights, the algorithm calculates the threshold value. Each onset cube needs to have the variable weights sum greater or equal than the threshold value.

Each variable in the ISOP expression is replaced by its respective weight, assigned by the method. For each cube, the sum of the weight values is made and the lowest sum value calculated is the threshold value.
In order to evaluate the algorithm efficiency, it is verified the number of functions the proposed algorithm can identify for each number of inputs, and these results are shown in Table 1. The enumeration of all TLF and of all NP-class representative identified TLF up to eight variables was calculated by Muroga [12].

### Table 1: Number of NP classes identified TLF

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>92</td>
<td>84</td>
<td>92</td>
</tr>
<tr>
<td>6</td>
<td>994</td>
<td>728</td>
<td>900</td>
</tr>
<tr>
<td>7</td>
<td>28262</td>
<td>9221</td>
<td>16804</td>
</tr>
</tbody>
</table>

For up to four variables both methods identify all TLF. For five variables Palaniswamy method identifies 84 of the 92 functions while the proposed method identifies all functions. For six and seven variables Palaniswamy method identifies 73.2% and 32.6% of TLF respectively, whereas our approach identifies 90.4% of six variables functions and 59.4% for seven. The proposed algorithm is the first non-ILP method that is able to identify correctly all TLF up to 5 inputs and delivers superior results in comparison to the state-of-the-art algorithm.

The execution time is evaluated by calculating the average time spent to identify each function. All representatives TLF were evaluated by the proposed algorithm and the results are shown in Table 3. The values show that the algorithm is as fast as the other approach [11]. For all cases of one to seven variables, the average identified by function is less than half a millisecond. The execution time does not increase exponentially as the ILP algorithms, allowing use proposed algorithm in a synthesis flow. To verify the scalability of the proposed algorithm, random sets of 5000 threshold logic functions of eight to eleven variables were selected. The average execution time per function is calculated and the results are in Table 2.

### Table 2 - Execution time per identified TLF.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Time/function (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>0.4</td>
</tr>
<tr>
<td>7</td>
<td>0.6</td>
</tr>
<tr>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>9</td>
<td>0.7</td>
</tr>
<tr>
<td>10</td>
<td>0.8</td>
</tr>
</tbody>
</table>

V. CONCLUSIONS

A novel algorithm to identify and synthesize threshold logic functions is proposed. It uses inequalities generated from the truth table to guarantee the result is correct, but does not use linear programming. The weight assignment uses some of the inequalities as constrains and is performed in ascending order, defined by Chow parameters. These weights assigned are always the minimum, since uses a bottom-up approach to find the weights. The results show that this is the first non-ILP method that identifies all functions up five variables. Moreover, it identifies more functions than the other methods when the number of variables increases. The runtime per function is similar to the state-of-the-art approaches, and it is shown that the algorithm is scalable, enabling the application of the algorithm in a threshold logic circuit synthesis flow.

REFERENCES