# A Comparison of Ternary Logic Synthesis Methods for Static CNTFET Gates

Robert Bos
*1910728*
*r.bos1@student.tue.nl*

Luuk den Hartog
*1407619*
*l.d.hartog@student.tue.nl*

Tycho B.A. Smokers
*1242564*
*t.b.a.smokers@student.tue.nl*

***Abstract*—Following Moore's law, the ever decreasing transistor size in integrated circuits has introduced several quantum level effects that impose problems for the functioning of standard CMOS circuits. Ternary logic, implemented in Carbon Nanotube Field Effect Transistors (CNTFETs), poses a solution to this, by storing more information per transistor. In this paper, three different ternary logic synthesis methods for CNTFETs are compared in terms of transistor count for a ternary half adder, full adder, multiplier and several other 2-input MISO gates. The three different algorithms, that are evaluated, are based on different techniques: Ternary-Transformed Binary Decision Diagrams (TBDD), geometric interpretation of ternary functions, and a modified Quine-McCluskey algorithm. More specifically, several claims, made in previous research by Lee et al., are investigated. This is done by recreating the three techniques as algorithms in MATLAB. On top of that, a further improvement on the Quine-McCluskey algorithm is proposed. This proposed Quine-McCluskey algorithm performs significantly better in terms of transistor count than the other implemented algorithms, at the cost of being more computationally expensive. It results in an average reduction in the number of transistors of 37.6% for the gates analyzed in this paper. In contrast, the geometric algorithm performs worst of the researched synthesis techniques. The difference between the original Quine-McCluskey and TBDD based algorithm is insignificant and inconsistent over the examined gates. Further investigation using, e.g., SPICE simulations, is needed to ensure that the resulting circuits can be realized and are functionally correct. In addition, these simulations could be used to examine the performance of the algorithms with respect to other circuit parameters, such as power consumption and propagation delay.**

***Index Terms*—Electronic Design Automation, EDA, Ternary Logic, Logic Synthesis, Quine-McCluskey, TBDD, Geometric Representation, CNTFET.**

## I. Introduction

Since the invention of the first Integrated Circuit in 1958, the rapid increase in complexity has necessitated design automation methodologies and tools to be developed and improved. The electronic design process consists of many different abstraction layers. The highest abstraction layer, called the Electronic System Level (ESL), describes the behaviour of the system and analyses the correctness of it. One level below that, the system is described in high level components, such as registers or data processing units. This is called the Register-Transfer Level (RTL). Subsequently, the system is described at the gate level and the circuit level. This is where logic synthesis and optimization comes into play [1]. The goal of this optimization is to achieve a desired logical behaviour, i.e., a certain Boolean function, with the best possible physical implementation on gate level. What is considered optimal depends on the requirements of the specific system. However, most often, the optimal circuit contains the minimal amount of transistors, has the lowest power consumption and the lowest propagation delay.

To achieve this optimization, many different techniques have been developed over the years. This has been one of the driving factors behind the rapid increase in transistor concentration in ICs, following Moore's law. Because of its relevance, it has already been a topic of research for several decades. However, in recent years, the optimization of binary logic gates is reaching its limits. The downscaling of transistor sizes in integrated circuit has seen major improvement over the last decades. On the other hand, the interconnects between transistors and gates have not changed as much [2]. Because of this, the delay of the interconnects dominates the total delay of the logic gates. Making interconnects the critical limitation in IC optimization. In order to overcome this problem, multi-valued logic (MVL) poses a possible solution, because such logic allows for more information to be send through interconnects, as compared to binary logic. Since more information can be encoded in the same electrical signal. Overcoming the critical interconnect delay.

In this paper, ternary logic, which uses three logic levels, is discussed. Such ternary, or trinary, logic can be both balanced, i.e., containing values -1, 0 and 1, and unbalance, i.e., consisting of values 0, 1 and 2. In the Carbon Nanotube Field-Effect Transistors (CNTFETs) making use of ternary logic, the three unbalanced logic levels correspond to three voltage levels, 0 V, $V_{dd}/2$ and $V_{dd}$. These CNTFETs were first constructed in 1998, consisting of a single Carbon nanotube connected between two metal electrodes to form a field-effect transistor [3]. The introduction of these CNTFETs is driving the need for research into ternary logic synthesis. This is a relatively new research area, as compared to binary logic synthesis, which has been a topic of research for many decades. As a result of that, many binary logic optimization techniques have already been developed.

One of the oldest binary logic optimization techniques is the Karnaugh map. The Karnaugh map was first introduced in [4]. This method is used to simplify Boolean expressions by making use of the human ability to recognize patterns. In short, the method transfers the required outcome truth-table into a grid representation. Then several common outputs next to one another are lumped together. Based on this grouping a minimization is acquired for implementing a certain Boolean function. Because this method relies on manual pattern recognition, it is not suitable for automation.

Subsequently, the Quine-McCluskey method, which is functionally identical to Karnaugh mapping, was first presented in [5]. This method does not make use of a graphical representation, as used in Karnaugh mapping, but uses a tabular representation. This tabular form makes it easier to use

in computer algorithms. The technique itself consists of first finding all so called prime implicants of the Boolean function to be minimized. A prime implicant is a part of a Boolean function which cannot be minimized any further. After these steps, the essential prime implicants are to be found. These form the smallest combination of prime implicants required to implement the Boolean function. This combination of essential prime implicants results in a minimized expression for the Boolean function.

However, Karnaugh maps and other simple Boolean expression forms are not suitable for larger logic functions, since they grow exponentially with the number of input variables. To solve this, Lee came up with the Binary Decision Diagram (BDD) in 1958, [6], as a Boolean function representation in the form of a directed acyclic graph. It was popularized by Akers in 1978 [7]. Since then, BDDs have been a prevalent logic representation method. BDDs, however, also expand exponentially with the number of variables, making it important to identify sub-expressions to merge nodes and removed edges. Unlike a truth table, a BDD is not canonical unless it has been reduced to a so-called Reduced Ordered Binary Decision Diagram (ROBDD) where the Boolean variables are ordered. The logic synthesis is then performed by traversing the graph in this order to evaluate the Boolean function. There are several, exact and heuristic, ways to optimize BDDs for size and evaluation time of a Boolean function, which are often aimed at variable reordering. A common heuristic method for example is to choose the variable that occurs most frequently as the root of the BDD.

Another heuristic based method is the ESPRESSO algorithm [8], a two-level logic minimizer, similar to the Quine-McCluskey method but with a heuristic approach as opposed to an exact derivation [9]. Similar to the Quine McCluskey method, ESPRESSO makes use of prime implicants to form a table that is to be reduced using certain heuristics. This method does not necessarily produce a globally optimal output for a certain logic function to be implemented. However, it works much faster than an exact approach.

Similar to the BDD are the AND-Inverter Graph (AIG) and And-Or-Inverter graph (AOIG), introduced in [10]. These graphs work on the premise that any Boolean expression can be implemented by a combination of conjunction (AND), disjunction (OR), and inversion (INV) operations. After which subsequent optimization of these graphs is performed in order to obtain the best possible logic function implementation. Optimization of large A(O)IGs can be a very computationally expensive task. Therefore, new optimization methods are still being researched, such as in [11].

In addition to AOIGs, Majority-Inverter Graphs (MIGs) have fairly recently been a topic of research. Instead of using AND, OR, and INV operations, as done in AOIGs, it uses only majority (MAJ) and inversion (INV) operations. Any AOIG, and thus any AIG, can be represented by a MIG, since the MAJ operation can be seen as a generalization of the AND and OR operations. The optimization of a subsequent MIG is done by performing transformations, using the Boolean algebra proposed in [12], on it to create an equivalent, but better MIG in terms of the desired design parameters, such as area, delay and power. Most often, the optimization results in a reduction in the number of nodes.

Altogether, many binary logic synthesis and optimization techniques are available. However, their performance in new applications, such as multi-valued logic, is a relatively new topic of research. This paper will therefore review several ternary logic synthesis methods and compare their performance in minimizing the number of transistors to form logic gates.

TABLE I: Truth table of a two-input ternary AND gate.

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 2 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 2 | 0 | 0 |
| 2 | 1 | 1 |
| 2 | 2 | 2 |

The remainder of this paper is structured as follows. In Section II, the problem statement is explained. Section III presents the related work on the topics discussed in this paper. In Section IV, the design automation approach is described in detail, of which the results and their validation is discussed in Section V. Finally, the conclusions of this paper are presented in Section VI.

## II. Problem Statement

As explained in the previous section, several approaches exist for the synthesis and optimization of logic functions. In this paper, three different approaches for the optimization of ternary logic for CNTFET circuits are compared. Lee et al., [13] claim that their modified Quine-McCluskey (QMC) algorithm performs significantly better than the methods proposed prior to this paper in [14] and [15], which are based on a geometric interpretation of ternary functions (GEO) and Ternary-Transformed Binary Decision Diagrams (TBDD), respectively.

In this paper, these three approaches are replicated and compared, in order to validate the results of the paper by Lee et al. The design automation algorithms are implemented in MATLAB. These algorithms use an unbalanced ternary truth table as input, with values 0, 1 and 2. This truth table represents the desired output of the system to certain inputs. As an example, the truth table for a two-input ternary AND gate is given in Table I.

The algorithms automate the synthesis of the minimal ternary expression that corresponds to the desired input-output behaviour specified in the truth table. The exact form of this expression differs per algorithm. From this expression, the design of a circuit is found with the minimal number of transistors.

Lee et al. compared the three algorithms in terms of transistor count for a ternary half adder, full adder and multiplier, resulting in the following claims to be validated:

1) for a ternary half adder, the TBDD algorithm, [15], yields a 20.8% improvement in transistor count compared to the QMC algorithm, [13], and a 34.5% improvement compared to the GEO algorithm, [14];
2) for a ternary full adder, [15] yields a 9.4% improvement in transistor count compared to [13] and an 8.4% improvement compared to [14];
3) for a ternary multiplier, [13] yields a 6.3% improvement in transistor count compared to [15] and a 30.2% improvement compared to [14].

In [13], the algorithms are also compared in terms of average power consumption, worst-case propagation delay and power-

delay product, which results in the Quine-McCluskey algorithm being a significant improvement over the other two. However, these comparisons are out of the scope of this paper. Therefore, the transistor count for the synthesis of two other basic ternary gates is investigated, being the AND and OR gate, and the computation time of the algorithms is examined.

## III. Related Work

Multi-valued logic synthesis has been a topic of research for many years. Multi-valued logic can be represented in several ways similar to binary logic, such as the sum-of-products form, multi-valued Boolean networks (MV-networks) or Multi-Valued Decision Diagrams (MDDs) [16]. The authors of this paper show that the ESPRESSO-II algorithm for binary logic minimization can be extended for multi-valued logic minimization using the Multi-valued Shannon Expansion Theorem.

The specific form of multi-valued logic discussed in this paper, i.e., ternary logic, is often researched in combination with reversible logic synthesis, which poses a solution to the quantum effects observed in nanometer scale transistors and ICs. Such reversible computing, and thus reversible logic, is needed in the development of quantum computing [17]. In ternary quantum logic, the basic unit of information is called a qutrit, which can have one of three states $|0\rangle$, $|1\rangle$ or $|2\rangle$. Rani and Thangkhiew explain the fundamental terminologies, logic gates and synthesis approaches, e.g., the Ternary Decision Diagram (TDD), of such ternary reversible quantum logic in [18]. In earlier work, Mandal et al., [19], also describe the basics of ternary logic, such as the basic operations AND, OR and NOT, i.e.,

$$\text{AND}(A,B) = \begin{cases} A, & \text{if } A \leq B; \\ B, & \text{otherwise;} \end{cases} \tag{1}$$

$$\text{OR}(A,B) = \begin{cases} A, & \text{if } A \geq B; \\ B, & \text{otherwise;} \end{cases} \tag{2}$$

$$\text{NOT}(A) = (A+1) \ \textit{mod}\ 3, \tag{3}$$

where $A, B \in [0,1,2]$. Furthermore, they show the ternary projection operations, subject to commutativity, associativity and distributivity, similar to binary logic. In their work, Mondal et al. extend this ternary reversible logic to balanced circuits, i.e., containing states -1, 0 and 1 [20]. Similar to [19], they explain the synthesis of ternary reversible logic gates and show the design of several arithmetic circuits, such as a half adder, a full adder and a multiplier.

Another prominent research area is the relatively new subject of ternary logic synthesis for CNTFET applications. Kim et al. first proposed their modified Quine-McCluskey algorithm for optimizing ternary logic gates in [21]. The authors further specified their algorithm in [13]. Their logic synthesis method relies on the generation of pull-up/down tables to obtain minterm canonical expressions by applying the proposed algorithm. After this, the minterms are merged by an iterative optimization process, yielding the sum of products (SOP) for each table. These SOPs are then selected for optimal transistor count and minimum propagation delay. In a later paper, other authors joined Lee and Kim to use the modified Quine-McCluskey algorithm for both balanced and unbalanced ternary logic to design a low power CNTFET circuit with pass transistors [22].

As mentioned in Section II, Lee et al. compare their modified Quine-McCluskey algorithm to [14] and [15]. In [14], logic

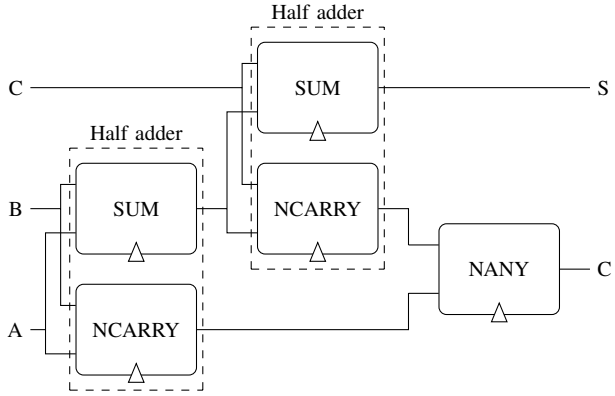


Fig. 1: Schematic of a ternary full adder, consisting of two ternary half adders and a NANY gate.

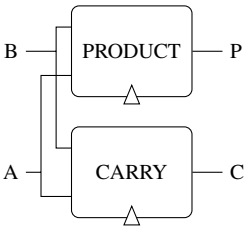


Fig. 2: Schematic of a ternary multiplier.

synthesis is performed using a graphical three dimensional (geometric) representation of the ternary function. This approach makes use of unary operators in combination with a scanning algorithm, to introduce unary operators in a specific order to reduce complexity. Resulting in a final SOP expression in terms of unary operators.

Vudadha et al., [15], proposed a Binary Decision Diagram (BDD) algorithm using 2:1 multiplexers claiming a striking 99% reduction in transistor count compared to the other most recent multiplexer based algorithms at the time. Although a BBD approach for three-value logic does not seem applicable, the methodology in [15] describes a transformation of a ternary function to a so-called Ternary-Transformed Binary Decision Diagram (TBDD). This allows the circuit implementation to be performed by replacing each node with 2:1 multiplexers instead of 3:1 multiplexers as used for a Ternary Decision Diagram (TDD).

Other researchers have also been investigating low power ternary logic circuits. In their 2009 paper, Lin et al. introduce an improved CNTFET ternary inverter design [23]. In later research, the gate complexity is increased and focus is shifted towards power optimization. Tabrizchi et al. [24] and Latha et al. [25] both present power optimized ternary logic gates using CNTFET technology. However, these studies focus more on transistor and circuit design, rather than logic synthesis and optimization.

## IV. Design Automation Approach

In the paper of Lee et al., [13], three types of circuits are introduced on which the synthesis methods are tested. These are the ternary half adder, full adder and multiplier. These three circuits consist of a combination of several 2 input gates, such as the SUM and the CARRY. The behaviour of all these gates is shown in (4)-(8). Additionally in Fig. 1 and 2 the combination of these basic gates for the ternary half adder, ternary full adder and

ternary multiplier are shown. In Lee et al. these three different circuits are used to make a comparison between a synthesis method making use of TBDDs [15], a geometric synthesis method [14], and their adjusted Quine-McCluskey method [13]. The remainder of this section explains the implementation of each method in detail.

$$\text{SUM}(A,B) = (A+B) \ mod\ 3; \tag{4}$$

$$\text{NCARRY}(A,B) = \begin{cases} 1, & \text{if } A+B \geq 3 \\ 2, & \text{otherwise;} \end{cases} \tag{5}$$

$$\text{NANY}(A,B) = \begin{cases} 0, & \text{if } A = B = 2 \\ 2, & \text{if } A \neq 2 \text{ and } B \neq 2 \\ 1, & \text{otherwise;} \end{cases} \tag{6}$$

$$\text{PRODUCT}(A,B) = (A \cdot B) \ mod\ 3; \tag{7}$$

$$\text{CARRY}(A,B) = \begin{cases} 1, & \text{if } A = B = 2 \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

### A. Ternary-Transformed Binary Decision Diagram (TBDD)

The TBDD approach relies on 12 propositions that provide guidance in building the binary decision diagram. 2:1 multiplexers and PTI or NTI gates are utilized for selection signals during the synthesis of this BDD at each node. While some propositions are supported by proofs, this section will omit these, since their validity has been verified in [15].

*Proposition 1* states that all TDDs can be transformed into (T)BDDs. In order to examine this, the Shannon expansion of a ternary function of a single output is given by

$$F = x^0 \cdot F_0 + x^1 \cdot F_1 + x^2 \cdot F_2, \tag{9}$$

where $F_0$, $F_1$ and $F_2$ represent the sub-graphs of the function. The signals $x^0, x^1$ and $x^2$ are mutually exclusive signals, in which the superscript does not represent a power. These signals are binary in nature, and take the values 0 or 2 according to

$$x^k = \begin{cases} 2, & \text{if } x = k \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Fig. 3 shows how a TDD can be decomposed. For every node, or input variable, an extra node is created. This expansion can be performed in three ways, each with different selection signals.

The complement of the selection signals in Fig. 3 represents the binary NOT as given by

$$\overline{x^k} = \begin{cases} 0, & \text{if } x = k \\ 2, & \text{otherwise.} \end{cases} \tag{11}$$

Since the decision tree only has values of 2 and 0, the graph is now very similar to that of a BDD. Table II displays the number of nodes in TDDs, BDDs, and TBDDs with $n$ input variables before any reduction. The number of nodes in the final decision diagram is important, as every node requires a mux, i.e., 2:1 in a BDD and 3:1 in a TDD. Therefore, it linearly attributes to the number of transistors, which is the most important design property that is assessed in this paper.

TABLE II: Number of nodes in the decision diagram, without reduction.

| | BDD | TDD | TBDD |
|---|---|---|---|
| #nodes | $2^n - 1$ | $\frac{3^n-1}{2}$ | $3^n - 1$ |

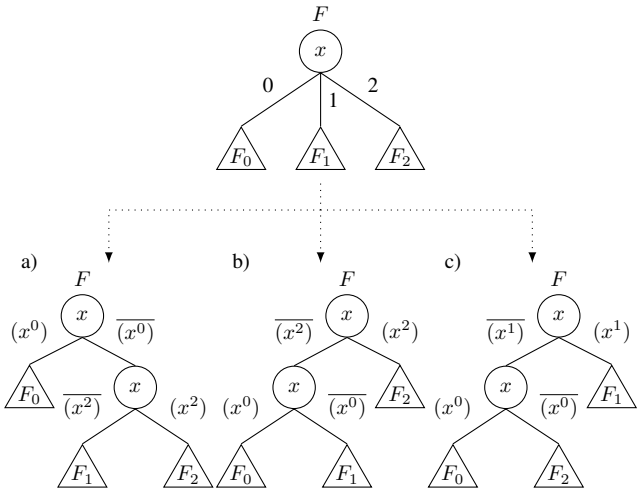


Fig. 3: Decomposition options to transform a TDD. Creating selection signals in option c) requires a ternary decoder.

As the selection signals for the variables are three valued, they need to be passed through Negative Ternary Inverter (NTI) and Positive Ternary Inverter (PTI) gates, of which the truth tables are given in Table III. For differentiating a value of 1, i.e., for $x^1$ and $\overline{x^1}$, however, a ternary decoder is needed. This increases the complexity, which is the reason the right-bottom graph, i.e., c), in Fig. 3 is not considered an option in [15].

The proposed synthesis algorithm aims to minimize the number of nodes by constructing the TBDD in a manner that enables maximal reduction. For example, if in Fig. 3 $F_1 = F_2$, then decomposing along 0 likely leads to less total nodes than decomposing along 2, as a node with equal outgoing edges can be removed or merged.

The remaining 11 propositions in [15] offer guidance in selecting the variable order and decomposition route, i.e., along $i = 0$ or along $i = 2$, starting with:

- *Proposition 2*: For a general TDD, if $F_1 = F_2$ then the TBDD represented by $x^0 \cdot F_0 + \overline{x^0} \cdot (\overline{x^2} \cdot F_1 + x^2 \cdot F_2)$, shown in Fig. 3 a), leads to optimal implementation;
- *Proposition 3*: For a general TDD, if $F_1 = F_2$ then the TBDD represented by $x^2 \cdot F_2 + \overline{x^2} \cdot (\overline{x^0} \cdot F_1 + x^0 \cdot F_0)$, shown in Fig. 3 b), leads to optimal implementation.

The next few propositions are based on the templates shown in Fig. 4:

- *Proposition 4, 5*: For a 1-input ternary function, applying the templates in Fig. 4 possible leads to optimal implementation. When the node represents a ternary function where the input is equal to the output, the node may be removed.

*Proposition 6* to *9* extend the previous propositions to a 2-input ternary function. However, since the algorithm does not utilize these transitions, the examples and steps are omitted here. They are merely an intermediate step towards the $n$-input ternary function. The final propositions for the $n$-input ternary function,

TABLE III: Truth tables for ternary inverter gates.

| $x$ | NTI($x$) | PTI($x$) |
|---|---|---|
| 0 | 2 | 2 |
| 1 | 0 | 2 |
| 2 | 0 | 0 |

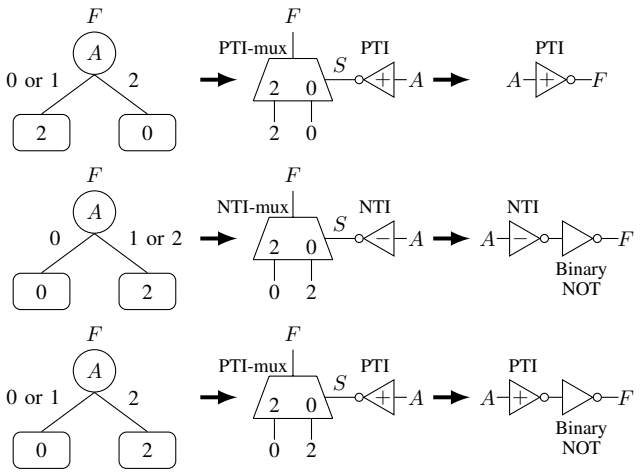


Fig. 4: 1-input ternary function TBDD templates. [15]

with input variables $x_i$ and ternary function $F(x_1, x_2, ..., x_n$, can be summarized as:

- *Proposition 10*: If $F_{x_i^0}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = F_{x_i^1}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = F_{x_i^2}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$, then decomposition along input $x_i$ leads to a reduced TBDD;
- *Proposition 11*: If $F_{x_i^0}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = F_{x_i^1}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ or $F_{x_i^1}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = F_{x_i^2}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$, then decomposition along input $x_i$ leads to a reduced TBDD in terms of ($n$-1)-input functions;
- *Proposition 12*: If $F_{x_i^0}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ or $F_{x_i^2}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ is a constant function, i.e., always gives the same outcome, then decomposition along input $x_i$ leads to a reduced TBDD in terms of ($n$-1)-input functions.

The algorithm described in [15] decomposes an $n$-input function into multiple ($n$-1)-input functions iteratively, until only 1-input functions remain. However, the provided pseudo-code is incomplete and implementinterg it in MATLAB did not produce the desired outputs. Consequently, a similar algorithm is implemented in MATLAB based on Algorithm 1. Lines 7-20 are used to determine the best option for inserting the remaining variables into the graph using the propositions. checkPropositions() is used to find whether any of the propositions 10, 11, or 12 hold to choose the next node, $x_i$, and edge signal, either $x_i^0$ or $x_i^2$. If none of these hold, the most significant digit (MSD) is chosen. Using $x_i$, two nodes are added with each two outgoing edges, as can be seen in Fig. 3. The destination of these edges is either a ternary digit, a node or a table with one less input variable, determined by the decompose() function. When there are no tables in the TBDD with more than 1 input variable, the templates in Fig. 4 can be applied. To this end, fun_one_input() is used to complete the TBDD before applying the synthesis algorithm.

When synthesizing a reduced TBDD to a CNTFET circuit, each node is substituted with a 2:1 mux and the corresponding NTI and PTI gates replace the selection signals. Additional binary inverters are introduced by applying the templates shown in Fig. 4, whenever possible. The transistor count is given by

**Algorithm 1** TBDD construction

**Input:** Truth table as list $O = O_0, O_1, ..., O_{3^n-1}$
**Input:** Variable string list $X$ % Optional
**Output:** TBDD (reduced)
1: **if** $(nargin \leq 2)$ **then** % generate variable names
2: $X \leftarrow (x_1, x_2, x_3, ..., x_n, OUT)$
3: **end if**
4: $TBDD \leftarrow$ createEmptyGraph()
5: append($TBDD$, [X, O]) % Insert full table as top node
6: $one\_input\_lists\_only = 0$ % pre-TBDD is unfinished
7: **while** $(!one\_input\_lists\_only)$ **do**
8: $one\_input\_lists\_only = 1$
9: **for** all nodes **do**
10: $O' \leftarrow$get_table($node$)
11: $X' \leftarrow$get_vars($node$)
12: **if** (height($O'$)$> 3$) **then** % table has $\geq 2$ input
13: $one\_input\_lists\_only = 0$
14: [$x_i$, $signal$] $\leftarrow$checkPropositions()
15: $new\_lists \leftarrow$ decompose($O'$, $X'$, $x_i$ $signal$)
16: updateGraph($TBDD$, $x_i$, $signal$, $new\_lists$)
17: removeDuplicates($TBDD$)
18: **end if**
19: **end for**
20: **end while**
21: $TBDD \leftarrow$ fun_one_input($TBDD$)
22: **return** $TBDD$.

$$N_{trans} = 4 \cdot N_{mux} + 2 \cdot (N_{PTI} + N_{NTI} + N_{BINOT}). \quad (12)$$

### B. Geometric

The geometric approach is based on the ternary logic synthesis method presented in [14]. This method is called "geometric" because it makes use of a geometric interpretation of ternary functions. An example of the geometric representation of a three variable function is shown in Fig. 5. Where all possible input combinations are placed in three-dimensional space. These input combinations consists of $3^3 = 27$ points. With axes A, B and C corresponding to the inputs for variables A, B and C, respectively. This three dimensional representation can be decomposed into three, two-dimensional representations along an axis of choice. This results in three two-dimensional representations as the one shown in Fig. 6. This set of two-dimensional representations can be used to obtain a SOP expression for the desired logic function. In [14] this decomposition is implemented for ternary functions of up to 13 variables. However, the focus in this paper is on the logic synthesis of two-variable ternary functions. Therefore, variable decomposition in the geometric approach is not considered in this paper.

Ternary logic synthesis using the geometric approach makes use of so called unary operators. These unary operators are used in combination with the geometrical representation in order to obtain a SOP expression. unary operators are simple one input functions, with relatively simple circuit implementation [26]. In case of ternary logic, there are a total of 27 unary operators. In Table IV, the truth table of the unary operators used in this logic synthesis method is shown. However, not all input combinations are satisfied by these unary operators. The

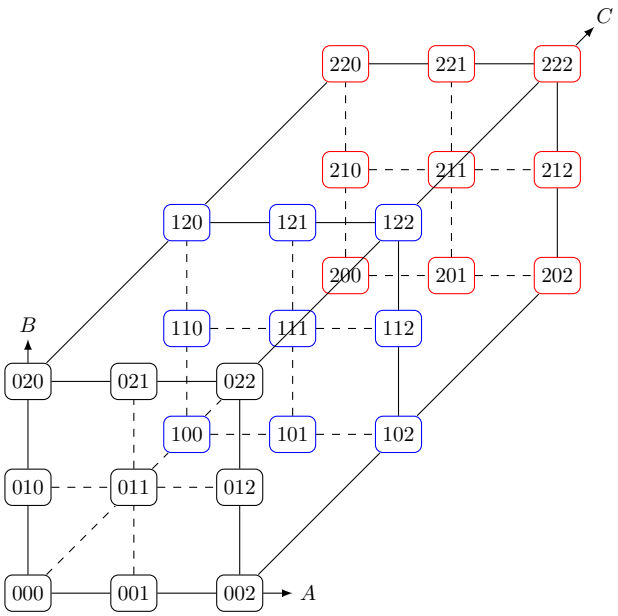


Fig. 5: Geometric representation of a three-variable ternary function, where 000 represents $a_0b_0c_0$.

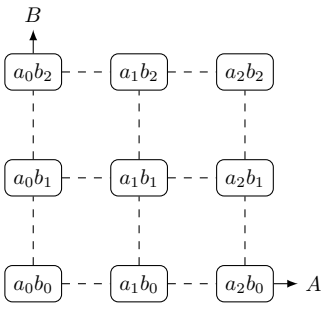


Fig. 6: Geometric representation of a two-variable ternary function, which is one layer of the cube in Fig. 5.

remaining combinations are made using a combination of these operators and basic AND and OR operations.

The main workflow of the geometric logic synthesis method starts with decomposing multi-variable functions into several two-dimensional representations, as the one shown in Fig. 6. Next, unary operators are introduced to satisfy the truth table in those two-dimensional representations, choosing the least amount of unary operators needed and the operators with the lowest implementation cost. Lastly, these two-dimensional representations are all fused into a circuit implementation using ternary multiplexers to implement the desired ternary function.

In order to choose unary operators to implement ternary functions, the two-dimensional representation is scanned and the best implementation of unary operators is sought. The act of "scanning" is explained in greater detail in Algorithm 2, later in this section. For now, to paint the picture of assigning unary operators, the AND ternary function will be taken as an

TABLE IV: Truth table of Unary operators [14].

| $A$ | $A_p$ | $A_n$ | $\overline{A}$ | $A^1$ | $A^2$ | $A_0$ | $A_1$ | $A_2$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 2 | 1 | 2 | 2 | 0 | 0 |
| 1 | 2 | 0 | 1 | 2 | 0 | 0 | 2 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |

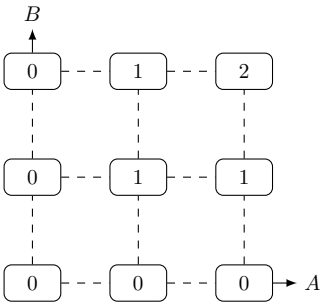


Fig. 7: Geometric representation of the ternary AND function.

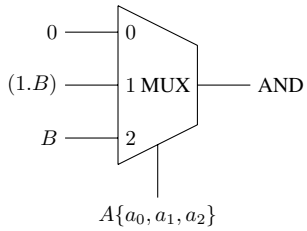


Fig. 8: Circuit implementation of ternary AND function using the geometric approach with Unary operators.

example. In Fig. 7, the two dimensional representation of the AND function is shown. First, a common signal is established. This is done by examining in what direction the non-zero entries are present, being either the horizontal direction or perpendicular to the B axis. This means $b_0$, $b_1$ or $b_2$ is identified as the common signal depending on which of the three rows holds more non-zero entries. Alternatively, if these non-zero entries are present mainly in the vertical direction or perpendicular to the A-axis, then $a_0$, $a_1$ or $a_2$ is identified as the common signal. After this common signal is identified a unary operator is paired with this signal to satisfy the two-dimensional representation of that specific row or column. This results in one minterm. Two-dimensional representations with multiple non-zero rows and/or columns result in a sum of products with multiple minterms.

For the case of the ternary AND function, the common signal can be established in either the horizontal or vertical direction. In this example, the vertical direction is chosen arbitrarily. However, in the actual algorithm both directions are compared in terms of transistor count. Consequently, the common signals are $a_1$ and $a_2$. For the column of $a_1$, the unary operator [011], from bottom to top, is needed. This is a combination of operators as $(1.B)$. On the other hand, for the column of $a_2$, the unary operator of [012], from bottom to top, is needed. This is operator $B$. As a result, the sum of products expression for the ternary AND function is given by

$$a_1.(1.B) + a_2.B. \tag{13}$$

As stated previously, the circuit implementation is realized by fusing all two-dimensional representations together using ternary, i.e., 3:1, multiplexers. These multiplexer take the unary operators, identified earlier in this section, as input. The input to the signal selection is the identified common signal. In Fig. 8, the circuit implementation for the ternary AND function is shown.

In Algorithm 2 the implemented solution for geometric ternary synthesis is shown. As input to the algorithm, the truth

**Algorithm 2** Geometric ternary synthesis

**Input:** Truth table of two variable ternary logic function as array $(x_0, x_1 ... x_{26})$

**Output:** function (Y) Sum of Products expression in terms of Unary operators

1: Assign the values of input $A$ and $B$ and output $X$ according to the truth table
2: Determine the values of $V_i$ and $H_i$ as per Table V
3: **if** $\sum_{i=0}^{8} X_i = 0$ **then**
4: $Y = 0$ % Output is a zero function
5: **else**
6: **if** ($X = A$ or $X = B$ or $X = 1$ or $X = 2$) **then**
7: $Y = A$ or $Y = B$ or $Y = 1$ or $Y = 2$ % Output is a constant function or equal to one of the inputs
8: **else if** ($H_0 = 0$ or $H_1 = 0$ or $H_2 = 0$) and ($V_0 = 0$ or $V_1 = 0$ or $V_2 = 0$) **then**
9: **if** $(H_0 + H_1 = 0)$ or $(H_1 + H_2 = 0)$ or $(H_2 + H_3 = 0)$) **then**
10: Choose one Unary operator along the horizontal direction.
11: **else if** $(V_0 + V_1 = 0)$ or $(V_1 + V_2 = 0)$ or $(V_2 + V_3 = 0)$) **then**
12: Choose one Unary operator along the vertical direction.
13: **else**
14: Choose two sets of two Unary operators along the vertical and Horizontal direction.
15: **end if**
16: **else**
17: **if** ($H_0 = 0$ or $H_1 = 0$ or $H_2 = 0$ or $V_0 = 0$ or $V_1 = 0$ or $V_2 = 0$) **then**
18: **if** ($H_0 = 0$ and $H_1 = 0$) or ($H_1 = 0$ and $H_2 = 0$) or ($H_0 = 0$ and $H_2 = 0$) or ($V_0 = 0$ and $V_1 = 0$) or ($V_1 = 0$ and $V_2 = 0$) or ($V_0 = 0$ and $V_2 = 0$) **then**
19: Choose one Unary operator along horizontal or vertical direction.
20: **else**
21: Choose two unary operators along the horizontal or vertical direction.
22: **end if**
23: **else**
24: Choose two sets of three Unary operators along the horizontal and vertical direction.
25: **end if**
26: **end if**
27: **end if**

table of a two-variable ternary function is given, which is assigned to the variables A and B for the input and X for the desired output (line 1). Firstly, several scenarios are checked such as zero functions or constant functions (lines 3-4), after which the scanning procedure starts (lines 5-27). This procedure makes use of the sum of the rows and columns in the two-dimensional representations as defined in Table V. This scanning checks to see where the empty rows or columns are located and assigns unary operators for the eventual SOP expression. This

TABLE V: Definition of horizontal and vertical summation $H_i$ and $V_i$.

| Summation | Definition |
|---|---|
| $H_0$ | $a_0b_0 + a_1b_0 + a_2b_0$ |
| $H_1$ | $a_0b_1 + a_1b_1 + a_2b_1$ |
| $H_2$ | $a_0b_2 + a_1b_2 + a_2b_2$ |
| $V_0$ | $a_0b_0 + a_0b_1 + a_0b_2$ |
| $V_1$ | $a_1b_0 + a_1b_1 + a_1b_2$ |
| $V_2$ | $a_2b_0 + a_2b_1 + a_2b_2$ |

introduction of unary operators is done in a specific order so as to reduce the complexity of the algorithm. After the sum of products expression in terms of unary operators is obtained, a transistor count is determined based on the amount of terms, the used unary operators and the necessary multiplexers.

For the example of the AND ternary function, as shown in Fig. 8, one ternary 3:1 multiplexer is used, which requires 18 transistors [14]. As an input, the unary operators $B$ are used, requiring no transistors. Due to the fact that it is just the $b$ input and the $(1.B)$ operator, which requires 5 transistors [14], the resulting transistor count is 23 for the AND function.

### C. Quine-McCluskey

The ternary Quine-McCluskey algorithm uses the ternary gate structure as shown in Fig. 9. In this structure, two paths are used to create the three possible output states, being 0 V, $\frac{1}{2}V_{DD}$ and $V_{DD}$. Firstly, the $V_{DD}$/GND path is used to create the logic 0 and 2. The logic 1 is then created by the half $V_{DD}$ path. From the truth table of a specific gate, a pull-up and pull-down table is constructed, which represents the states, ON or OFF, of the pull-up and pull-down networks in order to create the desired output. Table VI shows the pull-up/down table for a ternary AND gate. As can be seen in the table, several states are marked "X". In these cases, the state of the specific pull-up or pull-down network does not affect the output. These so-called don't care terms can be assigned any value. The assigned value changes the outcome of the Quine-McCluskey algorithm. Therefore, the don't care terms can be optimized to acquire the desired gate behaviour with the least amount of transistors. This don't care optimization is further explained later in this section.

From the pull-up/down table, with the don't care terms given a value of ON or OFF, the minterms can be extracted. These minterms represent the input combinations that cause a pull-up

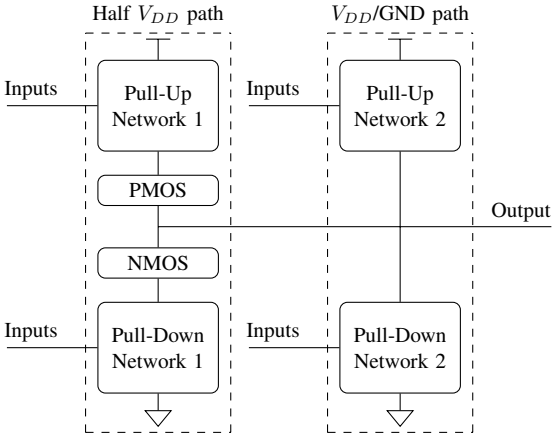


Fig. 9: Structure of a ternary gate. [21]

TABLE VI: Pull-up/down table for a ternary AND gate.

| $A$ | $B$ | $Y$ | $V_{DD}$/GND | | Half $V_{DD}$ | |
|---|---|---|---|---|---|---|
| | | | $U_{0/2}$ | $D_{0/2}$ | $U_1$ | $D_1$ |
| Before don't care process | | | | | | |
| 0 | 0 | 0 | OFF | ON | OFF | X |
| 0 | 1 | 0 | OFF | ON | OFF | X |
| 0 | 2 | 0 | OFF | ON | OFF | X |
| 1 | 0 | 0 | OFF | ON | OFF | X |
| 1 | 1 | 1 | OFF | OFF | ON | ON |
| 1 | 2 | 1 | OFF | OFF | ON | ON |
| 2 | 0 | 0 | OFF | ON | OFF | X |
| 2 | 1 | 1 | OFF | OFF | ON | ON |
| 2 | 2 | 2 | ON | OFF | X | OFF |
| Quine-McCluskey ( [13]) | | | | | | |
| 0 | 0 | 0 | OFF | ON | OFF | OFF |
| 0 | 1 | 0 | OFF | ON | OFF | OFF |
| 0 | 2 | 0 | OFF | ON | OFF | ON |
| 1 | 0 | 0 | OFF | ON | OFF | OFF |
| 1 | 1 | 1 | OFF | OFF | ON | ON |
| 1 | 2 | 1 | OFF | OFF | ON | ON |
| 2 | 0 | 0 | OFF | ON | OFF | OFF |
| 2 | 1 | 1 | OFF | OFF | ON | ON |
| 2 | 2 | 2 | ON | OFF | ON | OFF |
| Quine-McCluskey (proposed) | | | | | | |
| 0 | 0 | 0 | OFF | ON | OFF | ON |
| 0 | 1 | 0 | OFF | ON | OFF | ON |
| 0 | 2 | 0 | OFF | ON | OFF | ON |
| 1 | 0 | 0 | OFF | ON | OFF | ON |
| 1 | 1 | 1 | OFF | OFF | ON | ON |
| 1 | 2 | 1 | OFF | OFF | ON | ON |
| 2 | 0 | 0 | OFF | ON | OFF | ON |
| 2 | 1 | 1 | OFF | OFF | ON | ON |
| 2 | 2 | 2 | ON | OFF | ON | OFF |

or pull-down network to be ON. For Table VI, following the Quine-McCluskey algorithm of [13], the minterms are

$$U_{0/2} = A_2B_2, \tag{14}$$
$$D_{0/2} = A_0B_0 + A_0B_1 + A_0B_2 + A_1B_0 + A_2B_0, \tag{15}$$
$$U_1 = A_1B_1 + A_1B_2 + A_2B_1 + A_2B_2, \tag{16}$$
$$D_1 = A_0B_2 + A_1B_1 + A_1B_2 + A_2B_1. \tag{17}$$

These minterms, also called implicants, are assigned to a group, based on the amount of 1's and 2's in the implicant. In other words, e.g., the implicant $A_0B_1$ is assigned to group [1,0] and the implicant $A_2B_1$ is assigned to group [1,1]. These groups are then merged if they are adjacent, i.e., group [$x$,$y$] is adjacent to groups [$x$,$y$-1], [$x$,$y$+1], [$x$-1,$y$], [$x$-1,$y$-1], [$x$-1,$y$+1], [$x$+1,$y$], [$x$+1,$y$-1] and [$x$+1,$y$+1]. Next to their groups being adjacent, the implicants cannot differ by more than one trit, i.e., they have either the same value for $A$, or for $B$. As an example, the implicants $A_0B_0$ and $A_0B_1$ can be merged to $A_0(B_0 + B_1)$ in group [1,0]. The implicants that cannot be merged, are specified as prime implicants.

To derive the SOP, the merged implicants and prime implicants are presented in the form of a PIChart, with the decimal numerical value of the minterms on the x-axis and the implicants on the y axis, as shown in Fig. 10 for the $V_{DD}$/GND pull-down network, i.e., $D_{0/2}$, of a ternary AND gate. From this chart, all combinations of implicants are found that represent the decimal values of all minterms. In the example in Fig. 10, there is only one possible combination. The combinations are the SOPs which represent the behaviour of the gate. In case multiple possible SOPs are found, the SOP with the least number of transistors is chosen. In case multiple SOPs have the same amount of transistors, the SOP with the least number of transistors with chirality vector (10,0) is favoured, since it has

| | 0 | 1 | 2 | 3 | 6 |
|---|---|---|---|---|---|
| $A_0(B_1 + B_2)$ | | * | * | | |
| $(A_0 + A_1)B_0$ | * | | | * | |
| $(A_1 + A_2)B_0$ | | | | * | * |

Fig. 10: PIChart for the $V_{DD}$/GND pull-down network of a ternary AND gate.

a lower propagation delay [13]. For a ternary AND gate, this results in the SOPs given by

$$U_{0/2} = A_2B_2, \tag{18}$$
$$D_{0/2} = A_0(B_1 + B_2) + (A_0 + A_1)B_0 + (A_1 + A_2)B_0, \tag{19}$$
$$U_1 = A_1(B_1 + B_2) + A_2(B_1 + B_2), \tag{20}$$
$$D_1 = (A_1 + A_2)B_1 + (A_0 + A_1)B_2. \tag{21}$$

The amount of transistors, corresponding to a certain SOP, is based on [21]. The pseudo code for the Quine-McCluskey algorithm and the post-optimization of SOPs is given in Algorithm 1 and Algorithm 2 in [13], respectively.

As mentioned in the beginning of this section, the don't care terms of the pull-up/down table are optimized. The optimization algorithm is given in Algorithm 3. In this optimization, all possible combinations of don't care terms are used to compute their respective optimal SOP (lines 3-6) and count the amount of transistors needed for it. The best performing combination of don't care terms, $OptDC$, is selected as the output (lines 7-11). The optimization can be considered a brute force approach, which is feasible for two input gates. However, for larger gates, the computation time will become a limitation.

At this point, the authors of [13] consider their algorithm finished. However, based on [21], some improvements can still be made. Therefore, three improvements to the algorithm are proposed. First of all, if all possible combinations of a trit are found as minterms, e.g., $A_0B_0$, $A_1B_0$ and $A_2B_0$, these minterms can be replaced by a single $B_0$. This single trit holds the numerical values of the three minterms, but can be represented by only a single transistor in the circuit implementation. This principle also holds for the merged implicants, where, e.g., $A_0(B_1 + B_2)$, $A_1(B_1 + B_2)$ and $A_2(B_1 + B_2)$ can be replaced by $(B_1 + B_2)$.

**Algorithm 3** Don't Care Optimization

**Input:** pull-up or pull-down table PT
**Output:** optimal don't cares DC for PT
1: $DCs \leftarrow$ FindDontCares($PT$)
2: $OptDC = DC_1$
3: **for all** $DC_i \in DCs$ **do**
4: $PT_i \leftarrow$ GeneratePT($DCs_i$)
5: $SOPs_i \leftarrow$ QuineMcCluskey($PT_i$)
6: $OSOP_i \leftarrow$ PostOptimization($SOP_i$)
7: **if** CountTR($OptDC$) > CountTR($DC_i$) **then**
8: $OptDC = DC_i$
9: **else**
10: $OptDC = OptDC$
11: **end if**
12: **end for**

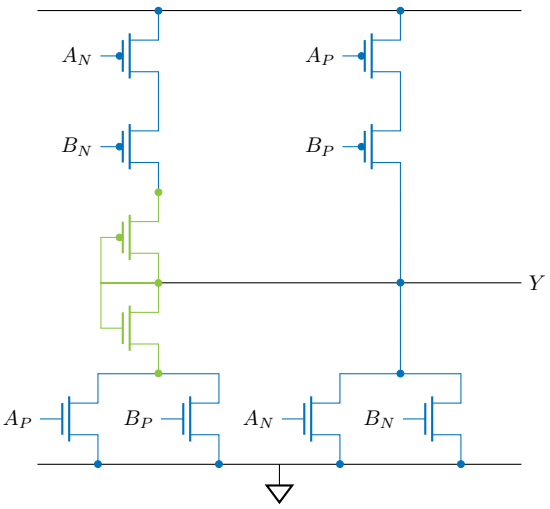


Fig. 11: Transistor level schematic of a ternary AND gate, following (22)-(25) and [21].

Secondly, two merged implicants can be merged again, based on the same set of rules. As an example, $A_1(B_1 + B_2)$ and $A_2(B_1 + B_2)$ can be merged together to form $(A_1 + A_2)(B_1 + B_2)$. This can also lead to a decrease in transistor count, according to [21]. Lastly, the PIChart as constructed in [13] only contains the prime implicants and merged implicants. However, by adding all implicants, other SOPs can be found, which can perform better in terms of transistor count.

Because of these proposed changes, the SOP optimization and don't care optimization yield different results, as compared to [13]. The resulting pull-up/down table of the proposed Quine-McCluskey algorithm is shown in Table VI. This proposed Quine-McCluskey algorithm yields the following SOPs:

$$U_{0/2} = A_2 B_2, \tag{22}$$
$$D_{0/2} = A_0 + B_0, \tag{23}$$
$$U_1 = (A_1 + A_2)(B_1 + B_2), \tag{24}$$
$$D_1 = (A_0 + A_1) + (B_0 + B_1). \tag{25}$$

These SOPs correspond to the transistor level schematic shown in Fig. 11, following [21].

## V. Validation

In this section the performance of the algorithms, described in the previous section, is discussed. The different algorithms are compared on the basis of the resulting transistor count for a certain logic function. Furthermore, a comparison is made between the implementation of the algorithms in this paper and their respective original papers. The MATLAB code is available at [27].

### A. Ternary-Transformed Binary Decision Diagram

The transistor counts for the gates, considered in this paper, are presented in Table VII. It can be observed that there are differences between the implementation of the algorithm and the source algorithm from [15]. Notably, the transistor count for the first five gates is different. One possible explanation for this discrepancy is that the source code was originally intended for large variable inputs, where the probability of a variable requiring a PTI gate, NTI gate, or binary NOT gate is small, and therefore negligible in the transistor count.

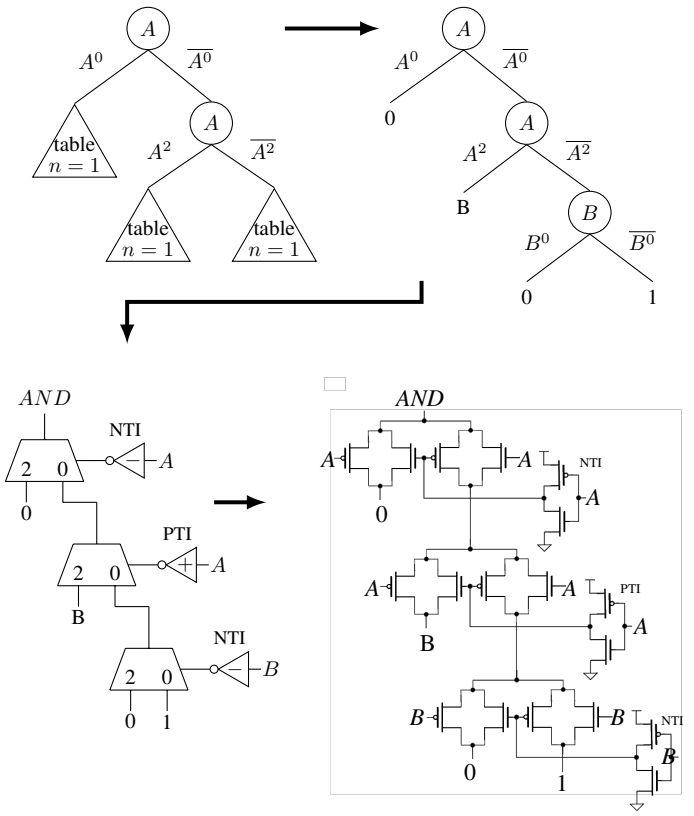


Fig. 12: Steps for implementing a ternary AND gate using the proposed algorithm based on [15], resulting in a total of 18 CNTFETs.

Fig. 12 shows how the new TBDD algorithm transforms the ternary AND function into a BDD and synthesizes it into a CNTFET circuit (the chirality vectors are omitted). However, to validate the new algorithm thoroughly, we need to perform HSPICE measurements to ensure that the circuit can be realized. Moreover, the ternary functions tested in this paper have a low variable count. In [15], ternary functions with as many as 13 inputs are tested for parameters such as transistor counts, number of 2:1 muxes and propagation delay. As the possibilities of realizing a ternary circuit increase with the variable count, it is expected that a better comparison can be made for large-input gates. The data generated in this paper could not directly be compared with the claims in [15], except for half adder and multiplier. Remarkably, the results of the source code in [15] seem to disagree with the claim in [15].

### B. Geometric

In Table VII the transistor count for the implementation of the geometric approach as well as the geometric implementation from the paper [14] is shown. In [13] the transistor count for the Half adder, Full adder and Multiplier implementation using the approach from [14] are listed. The geometric approach presented in this paper has, on average, a 25.2% higher transistor count than [14] for the three examined gates. The discrepancy is the most significant for the full adder. This is, among other things, due to the fact that the presented approach is tailored towards two-variable functions, which makes the transistor count for the full adder a summation of two half adders and a NANY gate. In contrast, the approach in [14] is made for functions with up to 13 input variables. Therefore, a full adder

TABLE VII: Comparison of optimization methods for several ternary logic gates. * denotes values that have been derived from the (adapted) source code and its transistor count equation, which is calculated as $N_{mux} \cdot 2 + 4 \cdot N_{inputs} + 4$.

| Method | Transistor count | |
|---|---|---|
| | Original paper | This paper |
| **AND** | | |
| TBDD ( [15]) | 24* | 18 |
| Geometric ( [14]) | | 23 |
| Quine-McCluskey ( [13]) | - | 20 |
| Quine-McCluskey (proposed) | - | 10 |
| **OR** | | |
| TBDD ( [15]) | 24* | 18 |
| Geometric ( [14]) | | 23 |
| Quine-McCluskey ( [13]) | - | 21 |
| Quine-McCluskey (proposed) | - | 13 |
| **SUM** | | |
| TBDD ( [15]) | 28* | 32 |
| Geometric ( [14]) | | 32 |
| Quine-McCluskey ( [13]) | - | 31 |
| Quine-McCluskey (proposed) | - | 28 |
| **NCARRY** | | |
| TBDD ( [15]) | 28* | 24 |
| Geometric ( [14]) | | 28 |
| Quine-McCluskey ( [13]) | - | 18 |
| Quine-McCluskey (proposed) | - | 10 |
| **NANY** | | |
| TBDD ( [15]) | 24* | 26 |
| Geometric ( [14]) | | 31 |
| Quine-McCluskey ( [13]) | - | 22 |
| Quine-McCluskey (proposed) | - | 12 |
| **PRODUCT** | | |
| TBDD ( [15]) | 24* | 24 |
| Geometric ( [14]) | | 25 |
| Quine-McCluskey ( [13]) | - | 25 |
| Quine-McCluskey (proposed) | - | 16 |
| **CARRY** | | |
| TBDD ( [15]) | 20* | 18 |
| Geometric ( [14]) | | 23 |
| Quine-McCluskey ( [13]) | - | 18 |
| Quine-McCluskey (proposed) | - | 6 |
| **Half Adder** | | |
| TBDD ( [15]) | 36, 44* | 50 |
| Geometric ( [14]) | 58 | 60 |
| Quine-McCluskey ( [13]) | 48 | 49 |
| Quine-McCluskey (proposed) | - | 38 |
| **Full Adder** | | |
| TBDD ( [15]) | 96, 100* | 138 |
| Geometric ( [14]) | 105 | 151 |
| Quine-McCluskey ( [13]) | 106 | 120 |
| Quine-McCluskey (proposed) | - | 88 |
| **Multiplier** | | |
| TBDD ( [15]) | 30, 40* | 42 |
| Geometric ( [14]) | 43 | 48 |
| Quine-McCluskey ( [13]) | 30 | 43 |
| Quine-McCluskey (proposed) | - | 22 |

is considered as a single gate, which makes for more room for optimization between the different sub-gates. However, further investigation is needed to fully explain the discrepancy between the implemented algorithm and the algorithm in [14].

### C. Quine-McCluskey

In order to validate the implementation of the Quine-McCluskey algorithm in this paper, compared to [13], the example given in [13] is used. For this ternary NCARRY gate, the algorithm's implementation in this paper produces the same SOPs as presented in [13]. Because this is the only given example, it is difficult to further validate the results for individual gates. For the three combinatorial gates discussed in this paper, i.e., a half adder, a full adder and a multiplier, the implementation of the Quine-McCluskey algorithm performs, on average, 19.5% worse than [13], in terms of transistor count. Further investigation is needed to determine the cause of the discrepancy between the implemented algorithm and the one from the original paper.

On the other hand, the proposed algorithm, with several improvements, results in an average reduction of 37.6% in the number of transistors for all the gates discussed in this paper, as compared to the implementation of the standard ternary Quine-McCluskey algorithm. It also performs 21.5% better, on average, than [13] in terms of transistor count. This is due to the simplifications made to the SOPs, allowing for more extensive optimization of don't care terms. This results in fewer necessary transistors to implement the same ternary function.

TABLE VIII: Comparison of the execution time of logic gate synthesis for the different implemented approaches.

| Method | Runtime for gate synthesis [s] | | |
|---|---|---|---|
| | Half adder | Multiplier | Full adder |
| TBDD | 0.001859 | 0.006836 | 0.006479 |
| GEO | 0.004770 | 0.008151 | 0.005398 |
| QMC | 3.384 | 20.38 | 4.210 |
| QMC (optimized) | 20.23 | 132.8 | 22.25 |

### D. Overview

In Table VII as well as Fig. 13 the resulting transistor counts from the different algorithms are shown. In Fig. 13, it can be seen how the different implemented approaches in this paper perform with respect to each other. The QMC algorithm with the additional optimization proposed in this paper performs the best, while the geometric approach implemented in this paper performs the worst in terms of transistor count. The QMC algorithm and TBDD algorithm perform similarly with insignificant and inconsistent differences in transistor count for the gates discussed in this paper.

Returning to the three main claims from Lee et al. [13] mentioned in Section II, the implementation of the algorithms in this paper yield the following results:

1) for a ternary half adder, the implemented TBDD algorithm, yields a 2.04% increase in transistor count compared to the implemented QMC algorithm and a 20.0% decrease compared to the implemented GEO algorithm;
2) for a ternary full adder, the implemented TBDD algorithm, yields a 15.0% increase in transistor count compared to the implemented QMC algorithm and an 8.0% decrease compared to implemented geometric approach in terms of transistor count;
3) for a ternary multiplier, the implemented QMC algorithm yields a 2.33% increase in transistor count compared to implemented TBDD algorith and a 11.6% decrease compared to the implemented geometric approach.

From this, it can be concluded that the implementation of the algorithms, as presented in this paper, perform significantly differently than in [13]. A possible explanation for this, is the optimization methods used in this paper, which could be nondeterministic. Therefore, they can possibly produce locally optimal solutions, which can therefore differ from the original research. However, during the research for this paper, the algorithms consistently produced the same results. Because of all this, it is difficult to validate the claims made by Lee et al.

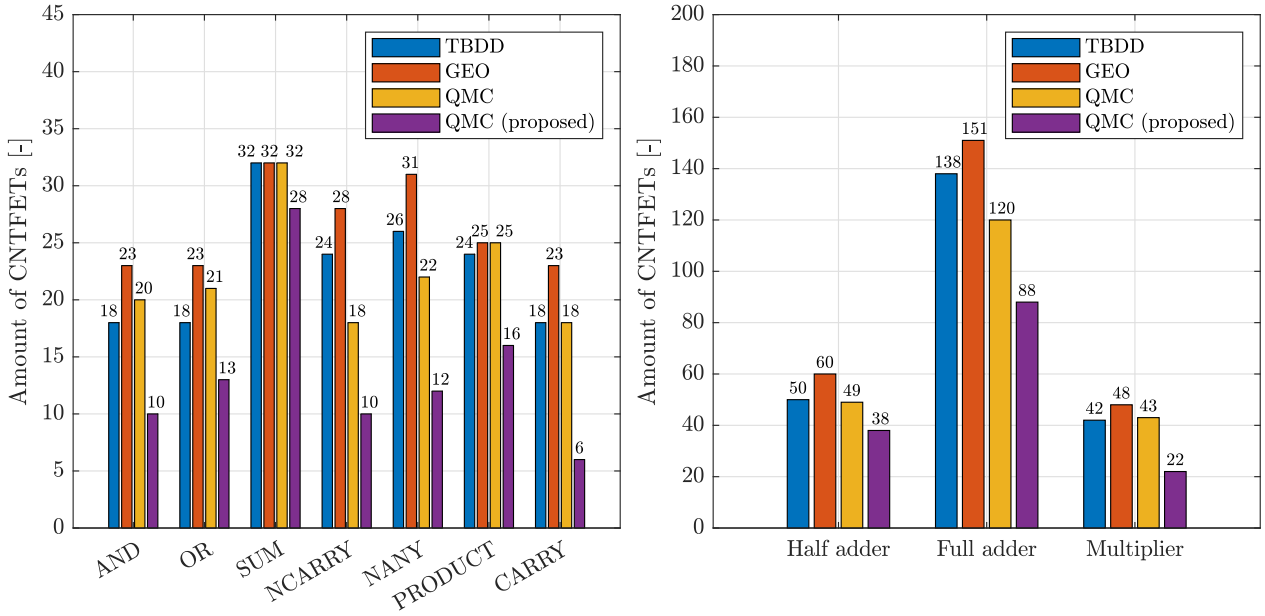


Fig. 13: Comparison of transistor count for several ternary logic gates, based on different optimization methods, being Ternary-transformed Binary Decision Diagrams (TBDD), Geometric interpretation (GEO) and Quine-McCluskey (QMC).

in [13].

In Table VIII a comparison between the execution times of the different implemented algorithms is shown for the synthesis of a half adder, full adder and multiplier. From this, it can be noted that the QMC implementation is significantly more computationally expensive than the TBDD and GEO implementations. Additionally, the optimized variant of the QMC implementation takes even more time. This is due to the brute force optimization of don't care terms and SOP generation. The combination of those optimizations leads to many different SOPs to be considered, which results in a high computation time.

Aside from comparing ternary logic synthesis methods in terms of transistor count, other variables from the circuit implementation such as Power-Delay Product can be considered, as is done in [13]. This is is a metric of circuit performance and energy consumption. In this paper, the comparison between synthesis methods was solely made in terms of transistor count. However, in future research it is important to take more aspects of the performance of the constructed circuit into account. This could be done by extracting the synthesized circuit from the algorithms and simulating it in a circuit simulator such as SPICE, resulting in performance metrics of the synthesized circuit. Additionally, in future research, the logic synthesis methods should be extended to handle more than two input variables, since this allows for more optimization of the overall circuit. For instance, larger gates can be constructed as a single gate with multiple inputs, instead of as a summation of several two-variable input gates.

## VI. Conclusions

This paper presents a comparison of three different ternary logic synthesis methods in terms of transistor count for several combinatorial circuits, including a ternary half adder, a full adder and a multiplier. The comparison is based on claims made by Lee et al. in [13]. The synthesis methods, discussed in this paper, comprise a Ternary-Transformed Binary Decision Diagram (TBDD) based algorithm, an algorithm based on the geometric interpretation of logic functions, and a Quine-McCluskey (QMC) algorithm. Furthermore, an extension to the Quine-McCluskey algorithm is proposed. The results show that the proposed QMC algorithm generates circuits with a significantly smaller number of transistors, as compared to the other algorithms. No significant difference is found between the TBDD and original QMC algorithm. The geometric algorithm performs considerably worse than the other algorithms, in terms of transistor count for the investigated logic gates.

More extensive circuit simulations need to be performed to validate the functionality of the generated circuits and test for other parameters, such as power consumption, propagation delay and the resulting power-delay product. In future research, larger input combinatorial circuits need to be analyzed to fully validate the claims in [13].

## Acknowledgements

This research was conducted as part of the course Electronic Design Automation at the Eindhoven University of Technology. We would like to thank prof.dr.ir. Jeroen P.M. Voeten for his guidance and supervision throughout the course.

## References

[1] J.-L. Huang, C.-K. Koh, and S. F. Cauley, "Chapter 8 - logic and circuit simulation," in *Electronic Design Automation*, L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, Eds. Boston: Morgan Kaufmann, 2009, pp. 449–512.

[2] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, 2001.

[3] S. J. Tans, A. R. M. Verschueren, and C. Dekker, "Room-temperature transistor based on a single carbon nanotube," *Nature*, vol. 393, pp. 49–52, 1998.

[4] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 72, no. 5, pp. 593–599, 1953.

[5] E. J. McCluskey, "Minimization of boolean functions," *The Bell System Technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.

[6] C. Y. Lee, "Representation of switching circuits by binary-decision programs," *The Bell System Technical Journal*, vol. 38, no. 4, pp. 985–999, 1959.

[7] Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, 1978.
[8] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algortithms for VLSI Synthesis*, 1st ed. Springer New York, 1984.
[9] S. Hassoun and T. Sasao, Eds., *Logic Synthesis and Verification*. Springer US, 2002. [Online]. Available: https://doi.org/10.1007/978-1-4615-0817-5
[10] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 3, pp. 198–223, 1963.
[11] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: a fresh look at combinational logic synthesis," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 532–535.
[12] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6.
[13] S.-Y. Lee, S. Kim, and S. Kang, "Ternary logic synthesis with modified quine-mccluskey algorithm," in *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*, 2019, pp. 158–163.
[14] B. Srinivasu and K. Sridharan, "A synthesis methodology for ternary logic circuits in emerging device technologies," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 8, pp. 2146–2159, 2017.
[15] C. Vudadha, A. Surya, S. Agrawal, and M. B. Srinivas, "Synthesis of ternary logic circuits using 2:1 multiplexers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4313–4325, 2018.
[16] R. Brayton and S. Khatri, "Multi-valued logic synthesis," in *Proceedings Twelfth International Conference on VLSI Design. (Cat. No.PR00013)*, 1999, pp. 196–205.
[17] A. N. Al-Rabidi, *Reversible Logic Synthesis*, 1st ed. Springer Berlin, 2003.
[18] P. M. Nesa Rani and P. Lyngton Thangkhiew, "A review on fundamentals of ternary reversible logic circuits," in *2020 International Conference on Computational Performance Evaluation (ComPE)*, 2020, pp. 738–743.
[19] S. B. Mandal, A. Chakrabarti, and S. Sur-Kolay, "Synthesis techniques for ternary quantum logic," in *2011 41st IEEE International Symposium on Multiple-Valued Logic*, 2011, pp. 218–223.
[20] B. Mondal, P. Sarkar, P. K. Saha, and S. Chakraborty, "Synthesis of balanced ternary reversible logic circuit," in *2013 IEEE 43rd International Symposium on Multiple-Valued Logic*, 2013, pp. 334–339.
[21] S. Kim, T. Lim, and S. Kang, "An optimal gate design for the synthesis of ternary logic circuits," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 476–481.
[22] S. Kim, S.-Y. Lee, S. Park, K. R. Kim, and S. Kang, "A logic synthesis methodology for low-power ternary logic circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 9, pp. 3138–3151, 2020.
[23] S. Lin, Y.-B. Kim, and F. Lombardi, "A novel cntfet-based ternary logic gate design," in *2009 52nd IEEE International Midwest Symposium on Circuits and Systems*, 2009, pp. 435–438.
[24] S. Tabrizchi, F. Sharifi, A.-H. Badawy, and Z. M. Saifullah, "Enabling energy-efficient ternary logic gates using cnfets," in *2017 IEEE 17th International Conference on Nanotechnology (IEEE-NANO)*, 2017, pp. 542–547.
[25] A. Latha, S. Murugeswaran, and G. Yamuna, "Power optimized ternary arithmetic logic circuit using carbon nano tube field effect transistor," in *2022 International Conference on Electronics and Renewable Systems (ICEARS)*, 2022, pp. 354–360.
[26] D. M. Miller and M. A. Thornton, "Multiple valued logic - concepts and representations," in *Synthesis Lectures on Digital Circuits and Systems*, 2008.
[27] L. d. Hartog, T. Smokers, and R. Bos, "Optimizing ternary logic using three different techniques," 2023. [Online]. Available: https://github.com/RobertB01/5SIB0-ternary-logic-synthesis