Compact Quantum Circuits for Autosymmetric and Dimension Reducible Functions

ANNA BERNASCONI, Department of Computer Science, University of Pisa, Italy
VALENTINA CIRIANI, Department of Computer Science, University of Milan, Italy
GIANMARCO CUCINIELLO, Department of Computer Science, University of Milan, Italy
NICOLAS MANINI, IMDEA Software Institute, and Universidad Politécnica de Madrid, Spain
ASMA TAHERI MONFARED, University of Bergamo, Italy

A quantum oracle is a reversible quantum subroutine that implements a classical Boolean function within a quantum algorithm. Classical synthesis methods for quantum oracles typically involve two steps: reversible logic synthesis and quantum compilation. In the reversible logic synthesis phase, producing a compact reversible circuit is crucial to minimizing the quantum cost of the final quantum circuit. In this paper, we investigate Boolean functions that simultaneously exhibit two distinct XOR-based regularities: autosymmetry and D-reducibility. These regularities can be effectively utilized for efficient reversible circuit synthesis of Boolean functions. In particular, we propose and implement a new method for the quantum synthesis of autosymmetric and D-reducible Boolean functions. The experimental results demonstrate the effectiveness of our approach on the considered benchmarks, showing reductions in T-count and CNOT-count after Clifford+T quantum compilation.

CCS Concepts: • **Theory of computation** → **Quantum computation theory**; • **Hardware** → *Reversible logic*; • **Computing methodologies** → Symbolic and algebraic manipulation.

Additional Key Words and Phrases: D-reducible functions, autosymmetric functions, reversible logic, quantum circuits

1 Introduction

In the last few years, technological enhancements in quantum architectures have led to renewed and growing interest in quantum computing, as well as the design of new secure cryptographic protocols. Consequently, research in quantum logic synthesis has also attracted considerable attention. In fact, many quantum algorithms, including Grover's search algorithm, usually require one to compute *oracles* [32], i.e., subroutines given as classical logic functions.

Standard approaches to synthesizing quantum oracles typically consist of two fundamental steps: reversible logic synthesis and quantum compilation. Since the evolution of quantum systems is governed by reversible unitary operators, classical logic functions must first be synthesized into reversible circuits. These circuits are then converted into quantum circuits by decomposing the reversible gates into sequences of basic quantum gates using a predefined gate library. These two steps should aim to reduce the total number of gates in the quantum

Authors' Contact Information: Anna Bernasconi, Department of Computer Science, University of Pisa, Pisa, Italy, anna.bernasconi@unipi.it; Valentina Ciriani, Department of Computer Science, University of Milan, Milan, Italy, valentina.ciriani@unimi.it; Gianmarco Cuciniello, Department of Computer Science, University of Milan, Milan, Italy, gianmarco.cuciniello@unimi.it; Nicolas Manini, IMDEA Software Institute, and Universidad Politécnica de Madrid, Madrid, Spain, nicolas.manini@imdea.org; Asma Taheri Monfared, University of Bergamo, Bergamo, Italy, asma.taherimonfared@unibg.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1550-4840/2026/9-ART
https://doi.org/10.1145/3845812

Just Accepted

circuits that will be executed. In recent times, several new approaches for reversible circuit synthesis and quantum compilation have been proposed in the literature. One such method explores various structural regularities of Boolean functions to synthesize compact quantum circuits, such as autosymmetry [3] and D-reducibility [8], which are based on the notion of affine spaces and expressed using the XOR operator. A study detailed in the literature [5] investigates the multiplicative complexity of autosymmetric and D-reducible functions. Furthermore, the experimental results indicate that approximately 9% of these regular functions exhibit both autosymmetry and D-reducibility.

In this paper, we propose and implement a strategy for the quantum synthesis of Boolean functions that exhibit both autosymmetry and D-reducibility. These functions are defined by their minterms being confined to an affine subspace smaller than the whole Boolean cube and expressed using the XOR operator [3, 6, 23]. These functions are prevalent in classical benchmarks; for instance, approximately 24% of the functions in the classical ESPRESSO benchmark suite [47] exhibit at least one autosymmetric output, and 70% exhibit a D-reducible output. Efficiently decomposing these regular functions in polynomial time makes them attractive targets for quantum synthesis, allowing for the development of circuits that are not only compact but also easier to implement.

In addition to the classical Espresso and LGSynth'89 suites, we also considered the more recent EPFL benchmark suite [1]. For this suite, the autosymmetry and D-reducibility tests were run with a timeout of 5 seconds per output. Of the outputs for which these tests completed within the timeout, 395, corresponding to approximately 18%, were found to satisfy both properties. The EPFL experiments reported in Section 4 are therefore performed on this subset.

In general, we expect structural regularities to occur more frequently in arithmetic functions, which arise in the implementation of arithmetic quantum oracles. Indeed, arithmetic circuits typically exhibit a significant amount of regularity, making them suitable targets for decomposition techniques based on autosymmetry and D-reducibility. In contrast, we expect such regularities to be much less common in the cryptographic components of quantum circuits, such as the S-boxes of symmetric ciphers and cryptographic hash functions. More specifically, while the linear layers of these constructions may still exhibit exploitable regular structures, their nonlinear cryptographic components are intentionally designed to behave as pseudorandom Boolean functions and, by definition, should not exhibit significant structural regularities. On the other hand, the study of quantum synthesis techniques for Boolean functions remains important regardless of the application setting. Boolean functions constitute fundamental building blocks of quantum oracles used in many quantum algorithms. For instance, in Grover's search algorithm, the oracle implements a Boolean function that marks the solutions to the search problem. Depending on the specific search problem, the corresponding Boolean function may or may not exhibit structural regularities. Consequently, understanding how to efficiently synthesize different classes of Boolean functions is an important step toward the design of compact and efficient quantum oracles.

The approach proposed in this paper exploits both autosymmetry and D-reducibility to design cost-efficient quantum circuits. By structurally decomposing Boolean functions that exhibit these regularities, we reduce the number of quantum gates required for their implementation, leading to more compact reversible circuits, optimized for quantum compilation. From a theoretical perspective, we show that any Boolean function, in the considered class, can be realized as a reversible circuit by introducing only three additional output lines: two lines for storing the intermediate results of the decomposition, in addition to the usual output line reserved for the target function. We then investigate and formalize the interplay between autosymmetry and D-reducibility within the oracle-synthesis pipeline, showing both theoretically and experimentally that the order in which these properties are exploited may produce different reversible circuit realizations, even when the induced functional mapping is identical.

After reversible synthesis, the obtained circuit must be translated into a set of elementary quantum gates. In this work, we consider the Clifford+T gate library, which is commonly used in quantum compilation. Within this library, the T gate is a single-qubit gate, while the CNOT gate is a two-qubit gate acting on a pair of qubits. These

Just Accepted

two types of gates are important for different reasons. The number of T gates, usually called T-count, is a relevant metric in fault-tolerant quantum computing (FTQC), where quantum circuits are protected against errors by means of error-correcting techniques. In this setting, T gates are usually more expensive than Clifford gates, and therefore reducing their number is an important objective [10, 19]. The number of CNOT gates is also relevant, especially for noisy intermediate-scale quantum (NISQ) devices. In these devices, two-qubit operations are often more difficult to implement reliably than single-qubit operations, and they can significantly affect the quality of the final circuit [30, 31, 35]. For these reasons, in this paper we evaluate the proposed synthesis strategies mainly in terms of T-count and CNOT-count after Clifford+T quantum compilation.

The experimental results obtained from the XAG (XOR-AND-Inverter Graph) representations show that the proposed decompositions can reduce the cost of the compiled quantum circuits. The gain, however, is not uniform over all instances. For this reason, we report the reductions in a cost-driven setting: for each metric, the decomposed implementation is retained only when it does not increase the cost with respect to the standard synthesis flow after Clifford+T compilation.

For a metric m and a decomposition strategy d, we define the non-negative-gain subset as

$$S_{m,d} = \{i \mid m_i^{\text{std}} > 0 \text{ and } m_i^d \leq m_i^{\text{std}}\},$$

and the corresponding aggregate gain as

$$G_{\text{agg}}(m,d) = \frac{\sum_{i \in S_{m,d}} m_i^{\text{std}} - \sum_{i \in S_{m,d}} m_i^d}{\sum_{i \in S_{m,d}} m_i^{\text{std}}} \cdot 100.$$

Thus, the aggregate gain is computed from summed costs over $S_{m,d}$, and is not the average of the per-instance gains.

On the Espresso and LGSynth'89 benchmark sets, the A+D strategy gives aggregate reductions of 27% in T-count and 44% in CNOT-count, with median per-instance gains of 23% and 50%, respectively. The D+A strategy gives aggregate reductions of 27% in T-count and 54% in CNOT-count, with median per-instance gains of 22% and 53%, respectively. On the EPFL subset, the aggregate T-count reduction is 21% for both strategies, while the aggregate CNOT-count reductions are 59% for A+D and 55% for D+A. In these four EPFL cases, the corresponding median per-instance gain is 0%.

The organization of this paper is as follows. Section 2 provides the preliminary definitions of D-reducible and autosymmetric functions, ESOP forms, reversible circuits, and quantum compilation. Section 3 details our proposed methodology for constructing reversible circuits for autosymmetric and D-reducible functions. Section 4 evaluates the proposed decomposition and presents experimental findings on benchmark sets. Finally, we conclude this work in Section 5.

2 Preliminaries

In this section, we discuss the definitions and properties of D-reducible and autosymmetric regular functions, which form the basis for our subsequent analysis. Moreover, we briefly introduce ESOP forms, reversible circuits, and quantum compilation. In the following sections, we show how we can design quantum circuits for the considered regular functions.

2.1 Dimension Reducible Functions

Here, we explore the concept of dimension reducibility (*D-reducibility*), a particular regularity based on *affine spaces* [12, 13], which has been detailed in [6]. Recall that a vector subspace V of the classical Boolean vector space $(\{0,1\}^n, \oplus)$ is a subset of $\{0,1\}^n$ containing the zero vector $\mathbf{0} = (00\ldots0)$, such that for each v_1 and v_2 in V we have that $v_1 \oplus v_2$ is still in V. We have the following:

Just Accepted

Definition 1. *Let V be a vector subspace of $(0,1^n,\oplus)$, and let $\alpha \in \{0,1\}^n$ be a Boolean point, or vector. The set $A = \alpha \oplus V = \{\alpha \oplus v \mid v \in V\}$ is an* affine space *over V with translation point α.*

Example 1. *In Figure 1, the set $A = \alpha \oplus V$ is an affine space over the vector space $V = \{000000, 001001, 010010, 011011, 100100,$
$101101, 110110, 111111\}$, with translation point $\alpha = 000001 \in 0,1^6$. Therefore, $A = 000001 \oplus V = \{000001, 001000, 010011,$
$011010, 100101, 101100, 110111, 111110\}$. The dimension of A is the dimension of the corresponding vector space V.*

A	V
$x_1x_2x_3x_4x_5x_6$	$x_1x_2x_3x_4x_5x_6$
0 0 0 0 0 1	0 0 0 0 0 0
0 0 1 0 0 0	0 0 1 0 0 1
0 1 0 0 1 1	0 1 0 0 1 0
0 1 1 0 1 0	0 1 1 0 1 1
1 0 0 1 0 1	1 0 0 1 0 0
1 0 1 1 0 0	1 0 1 1 0 1
1 1 0 1 1 1	1 1 0 1 1 0
1 1 1 1 1 0	1 1 1 1 1 1

Fig. 1. An *affine space A* and the corresponding vector space V.

Consider the points of V and A sorted in binary order. See, for example, Figure 1. In the vector space V of dimension k, rows are indexed from 0 to 2^k-1 and the points with indices $2^0, 2^1, 2^2, ..., 2^{k-1}$ form the *canonical basis* B_A of V. The *canonical translation point* α_A is the point of A with index 0. Generally, the canonical representation of an *affine space* is given by its canonical translation point and its canonical basis. In each canonical basis vector, the variable corresponding to the first 1-component from left is called *canonical variable*. Variables that are not canonical on the canonical basis are called *non-canonical variables*.

Example 2. *Consider the affine space A in Figure 1. We have that $\alpha_A = 000001$ and $B_A = \{001001, 010010, 100100\}$, where $v_1 = 001001$, $v_2 = 010010$, and $v_3 = 100100$. In v_1, the first 1-component from the left corresponds to x_3; in v_2, it corresponds to x_2; and in v_3, it corresponds to x_1. Therefore, the canonical variables are x_3, x_2, and x_1, while the non-canonical variables are x_4, x_5, and x_6.*

An *affine space* can be algebraically represented by a *pseudoproduct* consisting of an AND of XORs or literals [12]. There are several ways to express *affine space* characteristic functions as a pseudoproduct; among them, we use the *canonical expression* (CEX) [23].

The following definition describes how to derive the CEX expression from an *affine space* [12]:

- each non-canonical variable appears in exactly one XOR factor, and each XOR factor is composed of one non-canonical variable and possibly some canonical variables;
- the canonical variables that appear in the XOR factor corresponding to the non-canonical variable x are the canonical variables (if any) appearing with value 1 in the vectors of B_A where x is equal to 1;
- all the canonical variables are not complemented, a non-canonical variable is complemented in its XOR factor if and only if its corresponding component in α_A is 0.

Example 3. *Consider again the affine space A of Figure 1. Its canonical translation point is $\alpha_A = 000001$, and its canonical basis is $B_A = v_1, v_2, v_3 = 001001, 010010, 100100$. We derive one XOR factor for each non-canonical*

Just Accepted

variable. For $v_1 = 001001$, the first 1-component from the left identifies x_3 as the canonical variable, and the same basis vector also contains the non-canonical variable x_6. Since the x_6 component of α_A is equal to 1, x_6 is not complemented, and the corresponding factor is $x_3 \oplus x_6$. Similarly, $v_2 = 010010$ identifies x_2 as the canonical variable and x_5 as the corresponding non-canonical variable. Since the x_5 component of α_A is equal to 0, x_5 is complemented, yielding the factor $x_2 \oplus \bar{x}_5$. Finally, $v_3 = 100100$ identifies x_1 as the canonical variable and x_4 as the corresponding non-canonical variable. Since the x_4 component of α_A is equal to 0, the corresponding factor is $x_1 \oplus \bar{x}_4$. Therefore, the CEX expression for this affine space is $\chi_A = (x_3 \oplus x_6)(x_2 \oplus \bar{x}_5)(x_1 \oplus \bar{x}_4)$.

Building on the above definitions, we can now introduce *D-reducible* regular functions. Essentially, the points (or minterms) of *D-reducible* functions are entirely located within an *affine space* that is strictly smaller than the entire Boolean cube $\{0,1\}^n$. When a function f is classified as *D-reducible*, it can be expressed as $f = \chi_A \cdot f_A$, where A represents the smallest affine space containing f and is referred to as the associated *affine space* of f. Consequently, χ_A denotes the characteristic function of A, while f_A is the projection of f onto A.

In the following examples, we use Karnaugh maps only as a visual aid to show how the onset of a function is contained in an affine space and how the corresponding projection is obtained. A Karnaugh map is a graphical representation of the truth table of a Boolean function, where adjacent cells differ in the value of one variable only. For instance, for a two-variable function $g(x_1, x_2)$ with onset $\{01, 11\}$, the Karnaugh map contains two cells equal to 1 in the positions corresponding to $\{01\}$ and $\{11\}$, and the remaining cells are equal to 0.

Example 4. *The Karnaugh map on the left side of Figure 2 illustrates the D-reducible function $f = \{00011, 01010, 01110,$
$10001, 10101, 11000\}$. The affine space A associated with f may also contain points where the original function is equal to 0. In this example, the points 00111 and 11100 belong to A, but they are not in the onset of f. Therefore, they correspond to zero-valued points in the projected function f_A, and they are not treated as don't-care conditions. Points outside A are instead removed by the characteristic function χ_A.*

According to the Karnaugh map on the right side of the figure, the projected function f_A depends on three variables x_1, x_2, x_3. It should be noted that although the number of the onset minterms is the same for both f and f_A, they are compacted in a smaller space (map) in the f_A function. If we synthesize f and f_A in the classical SOP framework, we obtain that f and f_A equal to $\bar{x}_1 x_2 x_3 \bar{x}_4 + x_1 \bar{x}_2 \bar{x}_3 x_4 + \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 x_5 + x_1 x_2 \bar{x}_3 \bar{x}_4 \bar{x}_5$ and $\bar{x}_3 + x_1 \bar{x}_2 + \bar{x}_1 x_2$, respectively. As a result, the overall number of products is reduced from 4 to 3, and the overall number of literals is reduced from 18 to 5. Moreover, the canonical basis can be derived in polynomial time by exploiting Gauss-Jordan elimination, as described in [22]. Finally, a more compact form for the function f can be derived as $(x_1 \oplus x_4)(x_2 \oplus x_5)(\bar{x}_3 + x_1 \bar{x}_2 + \bar{x}_1 x_2)$, where $(x_1 \oplus x_4)(x_2 \oplus x_5)$ is the CEX representing the affine space χ_A.

2.2 Autosymmetric Functions

Autosymmetric functions exhibit another type of regularity based on affine spaces. Informally, the idea behind this concept is to call k-autosymmetric any Boolean function f depending on n variables that can be simplified to a smaller function f_k, depending only on $n-k$ variables. Following this idea, the degree of regularity of a Boolean function f is determined by its *autosymmetry degree* k, where $0 \leq k \leq n$. If $k = 0$, the function has no regularity. However, if $k \geq 1$, the function is identified as autosymmetric. Generally, an autosymmetric function f can be expressed through its *restriction function* f_k as follows:

$$f(x_1, x_2, \ldots, x_n) = f_k(y_1, y_2, \ldots, y_{n-k})$$

where variables $y_1, y_2, \ldots, y_{n-k}$ are XOR combinations of subsets of the original input variables x_i. These combinations are denoted as $\mathrm{XOR}(X_i)$, where $X_i \subseteq \{x_1, x_2, \ldots, x_n\}$. Equations

$$y_i = \mathrm{XOR}(X_i), \quad i = 1, \ldots, n-k$$

Just Accepted

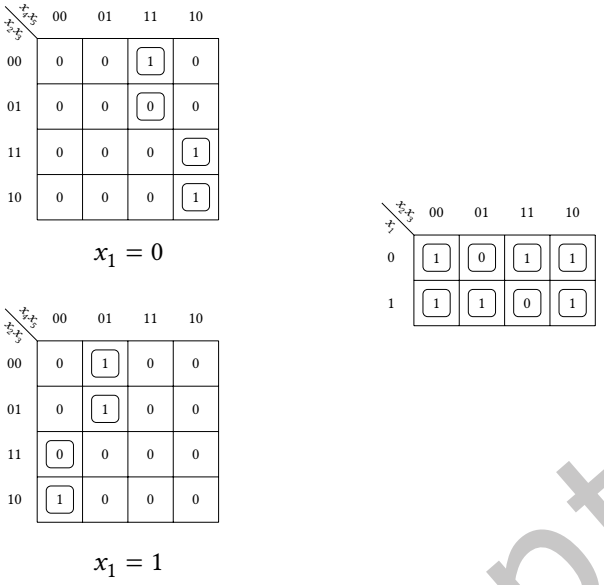


Fig. 2. Karnaugh maps of a D-reducible function f (left) and its corresponding projection f_A (right).

are called *reduction equations*. Thus, the autosymmetry test consists of finding the value k, the restriction f_k, and the reduction equations.

It is worth mentioning that autosymmetric functions depend, in general, on all their input variables, i.e., they are non-degenerate. However, degenerate functions that do not depend on all their variables are always classified as autosymmetric.

It is possible to synthesize the function f through its restriction f_k. This reconstruction requires an additional logic level of XOR gates, which take the original variables as inputs and outputs the new $n-k$ variables that serve as inputs to the circuit for f_k. The function f_k can typically be synthesized using different techniques for any logic minimization problem. For instance, in [4, 5, 9], an XAG representation of the restricted function has been derived to improve the evaluation of multiplicative complexity, and in [3] this representation has been used to design compact reversible and quantum circuits.

As demonstrated in [23], any k-autosymmetric function f has a corresponding k-dimensional vector space L_f, which contains all minterms α such that $f(x) = f(x \oplus \alpha)$ for all $x \in \{0,1\}^n$. The vectors in L_f are sorted by increasing binary order and indexed from 0 to 2^k-1. The subset of vectors indexed by $2^0, 2^1, \ldots, 2^{k-1}$ is termed the canonical basis B_L of L_f. There are k fully independent variables in L_f, which are known as *canonical variables* and assume all the possible combinations of 0, 1 values in the vectors of the vector space L_f. Conversely, *noncanonical variables* in L_f, which are not canonical, assume either a constant value or are formed as linear combinations of the canonical variables. In the canonical basis B_L of L_f, canonical variables are those corresponding to the first 1-component from the left. These variables are essential to define the function f in its compact form f_k.

The restriction f_k represents a more compact form of f, containing only $|f|/2^k$ minterms, where $|f|$ represents the total number of minterms in f. The reversible synthesis of f can be simplified to the synthesis of its restriction f_k, which can be determined in polynomial time in the dimension of some standard representations of the input function f, such as a Reduced Ordered Binary Decision Diagram (ROBDD) representation [7].

Just Accepted

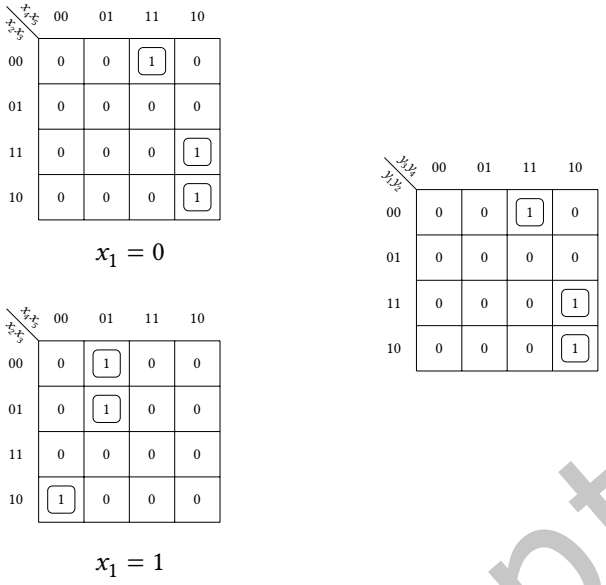


Fig. 3. Karnaugh maps of an 1-autosymmetric function f (left) and its corresponding projection f_k (right).

Example 5. *Figure 3 illustrates the Karnaugh map of the 1-autosymmetric function* $f = \{00011, 01010, 01110, 10001, 10101, 11000\}$ *with linear space* $L_f = \{00000, 11011\}$. *We can also determine the autosymmetry degree k by calculating* $\log_2 |L_f|$. *In this case,* $k = 1$, *which indicates that the function f is 1-autosymmetric and therefore all the points in the function f can be projected in the space* $\{0, 1\}^4$ *with respect to the reduction equations* y_1, y_2, y_3 *and* y_4. *In this example, the canonical basis is* 11011, *which designates* x_1 *as the canonical variable, while* x_2, x_3, x_4 *and* x_5 *are non-canonical variables. Therefore, we can obtain the reduction equations* $y_1 = x_1 \oplus x_2$, $y_2 = x_3$, $y_3 = x_1 \oplus x_4$, *and* $y_4 = x_1 \oplus x_5$. *The restricted function* f_1 *shows a simplified form of the original function f, having only 3 minterms, which are* $\{0011, 1010, 1110\}$, *as can be observed in the Karnaugh map on the right side of the figure.*

2.3 Autosymmetric and D-Reducible Functions

Two principal decomposition strategies are possible in the study of Boolean functions that exhibit both autosymmetry and D-reducibility. The first strategy involves beginning with D-reducibility, followed by exploiting autosymmetry. In contrast, the second strategy involves starting with autosymmetry and then proceeding to exploit D-reducibility. This paper presents quantum circuit synthesis specifically for these two approaches.

When employing the first strategy, the decomposition starts with D-reducibility. This involves expressing the Boolean function f as $f = \chi_A \cdot f_A$, where A represents an affine space and f_A is the projection of f onto A. Following this, the autosymmetry of f_A is used to obtain a representation $f = \chi_A \cdot f_{Ak}$, where f_{Ak} is defined based on the autosymmetric properties of f_A. This sequential approach ensures that the function f is represented in terms of both its affine space and its autosymmetric characteristics.

Example 6. *Consider the ongoing example. We apply D-reducibility followed by autosymmetry to the function. Consider the function f presented in Figure 2. As demonstrated in the Example 4, f is D-reducible, and its projection is* $f_A(x_1, x_2, x_3)$. *The set of minterms for* f_A *is given by* $f_A = \{000, 010, 011, 100, 101, 110\}$. *Next, we compute the*

Just Accepted

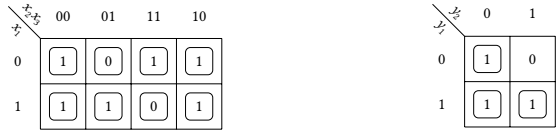


Fig. 4. Karnaugh maps of the function $f_A(x_1, x_2, x_3)$ (left) and of its corresponding projection $f_{A1}(y_1, y_2)$ (right).

autosymmetry decomposition for f_A. The Karnaugh map of f_A is also depicted on the left side of Figure 4, where it is observed that f_A exhibits autosymmetry. The corresponding vector space is $L_{f_A} = \{000, 110\}$, which has dimension $k = \log_2 |L_{f_A}| = 1$, indicating that f_A is 1-autosymmetric. Projecting the minterms onto the Boolean space $\{0, 1\}^2$ of the variables y_1 and y_2, we obtain the function $f_{A1}(y_1, y_2) = \{00, 10, 11\}$, which is represented in the Karnaugh map on the right side of Figure 4. The corresponding reduction equations are $y_1 = x_1 \oplus x_2$ and $y_2 = x_3$. A Sum-of-Products (SOP) form for f_{A1} is equal to $y_1 + \overline{y_2}$. Substituting the reduction equations, we obtain $y_1 + \overline{y_2} = (x_1 \oplus x_2) + \overline{x_3}$. Recalling that the characteristic function of A is $\chi_A = (x_1 \oplus x_4)(x_2 \oplus x_5)$, the original function f can now be written as

$$f(x_1, \dots, x_5) = (x_1 \oplus x_4) \cdot (x_2 \oplus x_5) \cdot [(x_1 \oplus x_2) + \overline{x_3}].$$

In contrast, the second strategy begins with autosymmetry. In this approach, we first determine the autosymmetry of the function f and obtain f_k, which is the restriction of f that reflects its autosymmetric property. Afterward, the D-reducibility of f_k is exploited, leading to the representation $f = \chi_A \cdot f_{kA}$, where A denotes the affine space and f_{kA} is the projection of f_k onto A. This method simplifies the function f by reducing its autosymmetric components before projecting it onto its affine space.

Example 7. *Let us consider the ongoing function again, this time applying the decomposition method where autosymmetry is performed first, followed by D-reducibility. Referring to Figure 3, Example 5 shows that f is 1-autosymmetric, and the restriction f_1 (shown in the Karnaugh map on the left side of Figure 5) corresponds to the set of minterms $f_1(y_1, y_2, y_3, y_4) = \{0011, 1010, 1110\}$ in $\{0, 1\}^4$. Next, we perform the D-reducibility decomposition on f_1. Projecting f_1 onto A, we derive the Boolean function $f_{1A}(y_1, y_2) = \{00, 10, 11\}$, which is illustrated in the Karnaugh map on the right side of Figure 5. The characteristic function for A is $y_3 \cdot (y_1 \oplus y_4)$. For simplicity, we represent f_{1A} in SOP form as $f_{1A} = (y_1 + \overline{y_2})$. It is worth noting that f_{1A} can be represented in various forms. In particular, we will use the XOR-And Graph (XAG) representation in the experimental section. Thus, we have*

$$f_1(y_1, y_2, y_3, y_4) = \chi_A \cdot f_{1A} = (y_3 \cdot (y_1 \oplus y_4)) \cdot (y_1 + \overline{y_2}).$$

To reconstruct the original function f, we substitute the variables y_1, y_2, y_3, y_4 with their corresponding reduction equations as computed in Example 5. This gives us

$$f(x_1, \dots, x_5) = [(x_1 \oplus x_4) \cdot ((x_1 \oplus x_2) \oplus (x_1 \oplus x_5))] \cdot [(x_1 \oplus x_2) + \overline{x_3}].$$

After simplification, the final expression becomes

$$f(x_1, \dots, x_5) = (x_1 \oplus x_4) \cdot (x_2 \oplus x_5) \cdot [(x_1 \oplus x_2) + \overline{x_3}].$$

Finally, it is worth noting that this decomposition is identical to the one obtained using the alternate strategy as we have previously shown in Example 6. Indeed, as proven in [9], for completely specified functions that are both autosymmetric and D-reducible, the two decomposition strategies are equivalent and provide the same decomposed function, i.e., $f_{Ak} = f_{kA}$. On the other hand, when the initial function f is incompletely specified, the

Just Accepted

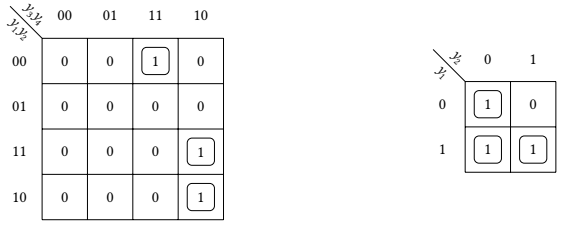


Fig. 5. Karnaugh maps of the function $f_1(y_1, y_2, y_3, y_4)$ (left) and of its corresponding projection $f_{1A}(y_1, y_2)$ (right).

decomposed functions f_{Ak} and f_{kA}, derived by applying different strategies for specifying don't care conditions, can be different [9].

To summarize, when a completely specified Boolean function is both D-reducible and autosymmetric, the two decomposition orders can be described as follows. In the D-reducibility–autosymmetry order, we first compute the affine space A containing the onset of f, express f as $f = \chi_A \cdot f_A$, and then apply autosymmetry to the projected function f_A, obtaining a further restricted function f_{Ak}. In the autosymmetry–D-reducibility order, we first compute the autosymmetry space L_f and the restriction f_k, and then apply D-reducibility to f_k, obtaining the projected function f_{kA} and its characteristic function. For completely specified functions, the final decomposed functions obtained by the two orders are equivalent, although the corresponding circuit constructions may still differ, as discussed in Section 3.

2.4 Relation with linear transformations

As a final observation, we note that both notions of autosymmetry and D-reducibility share similarities with recent work focused on reducing the number of variables required to represent partially defined classification functions [42, 43]. In both cases, the objective is to reduce the dimension of the space in which a Boolean function is represented and analyzed. In particular, the linear transformations introduced in [42, 43] resemble the reduction equations of autosymmetric functions, as both approaches define a new set of variables through linear (EXOR) combinations of the original ones. However, the underlying algebraic structures are fundamentally different. While autosymmetry is based on identifying affine subspaces over which the function is invariant, the variable reduction proposed in [42, 43] relies on general linear transformations and does not require the existence of affine invariance. Moreover, while the autosymmetry property always guarantees a reduction in the function's dimension, the converse does not hold: a reduction in dimension does not necessarily imply the existence of the autosymmetric property.

On the other hand, D-reducibility is more closely related to the notion of linear transformations. In both approaches, the objective is to identify a lower-dimensional space in which the target function can be represented and analyzed. In our framework, this space is the affine space A, which is associated with the *on-set* of the target function and is derived directly from its algebraic structure. By contrast, the approach proposed in [42, 43] searches for suitable linear transformations that reduce the dimension of the representation while preserving the information contained in the *care set* of the classification function, without relying on the specific algebraic properties exploited by D-reducibility. Furthermore, our framework applies to both completely and partially specified Boolean functions, whereas the approach in [42, 43] is specifically devoted to partially defined classification functions. This also suggests a possible integration of the two approaches: after the linear-transformation step, the resulting classification function could be further decomposed by exploiting autosymmetry and D-reducibility of its ON-set.

Just Accepted

2.5 ESOP forms

Boolean functions in Exclusive-or Sum-of-Products (ESOP) form are expressed using a series of AND gates at the first level and one multi-input XOR gate at the next level. ESOP forms continue to be of interest in research due to their important applications in various emerging technologies. Compared to standard SOP forms, ESOP forms provide several benefits, such as requiring fewer products to realize randomly generated functions [39, 40], more compact representations for arithmetic or communication circuits [41], higher testability properties [17, 20], and security [21, 29]. Furthermore, due to the inherent reversibility of the XOR operation [36], ESOP forms are essential for reversible logic circuit synthesis and quantum computing [15].

Recall that, for a Boolean function to be expressed in ESOP form, each off-set minterm must be covered by an even number of product terms involving the input variables, or none at all, whereas each on-set minterm must be covered by an odd number of these product terms.

A function can have multiple ESOP forms that are structurally different but semantically equivalent. The goal of ESOP minimization is to find the form with the lowest cost based on a given cost metric.

Example 8. *In Figure 2, the function f_A can be represented in ESOP form with two products and three literals as follows:*

$$ESOP(f_A) = \overline{x}_3 \oplus x_1\overline{x}_2x_3 \oplus \overline{x}_1x_2x_3$$

ESOP minimization is a computationally very hard problem that has garnered significant interest from algorithm designers due to its potential to reduce the overall costs of hardware realizations and software implementations. The exact problem of finding an ESOP with the minimum number of terms is NP-hard, and much more difficult than that of standard SOP minimization [36, 39]. No efficient method is known to obtain a minimum ESOP for a given function except for very small problems, which explains why practical approaches often rely on SAT- or search-based heuristics and exact methods are typically limited to small instances [28, 33, 34, 36, 38, 44].

Although ESOP forms can be quickly found through heuristic methods, only a subset of the possible search space is examined. Therefore, while these methods can produce small ESOP forms in a reasonable amount of time, they suffer from local minima, which are difficult to escape. On the other hand, while exact methods are guaranteed to find a minimal number of product terms, they can hardly handle more than 8 Boolean variables and a few product terms. In [36], a novel approach for exact synthesis based on Boolean satisfiability is presented, which is particularly fast and unaffected by the number of Boolean variables involved.

2.6 Reversible Circuits and Quantum Compilation

Reversible circuits ensure a one-to-one correspondence between each input vector and each output vector. Accordingly, the number of inputs equals the number of outputs. In these circuits, constant inputs are called *ancilla inputs*, while *garbage outputs* are produced to maintain one-to-one mapping, although their specific values are unimportant. Generally, reversible circuits are constructed from a sequence of Mixed-Polarity Multiple-Control (MPMC) Toffoli gates, which are able to implement any logic function.

We represent the MPMC Toffoli gates using the conventional notation, where $\oplus$ denotes the target line, while control lines are indicated by • for positive connections and ◦ for negative connections. If there are no control connections on the MPMC Toffoli gate, it operates as a NOT gate; with a positive control connection, it functions as a CNOT gate; and when there are only positive connections, it functions as a Multiple-Control Toffoli gate [32].

More formally, given a set of circuit lines $X = x_1, x_2, ..., x_n$, an *MPMC Toffoli gate* $T(C, t)$ has *control lines* $C = \{x_{j1}, x_{j2}, ..., x_{jc}\} \subset X$ and a *target line* $t \in X \setminus C$. The gate maps $t \rightarrow t \oplus (x_{j1}^{p1} \wedge x_{j2}^{p2} \wedge \ ... \ \wedge x_{jl}^{pl})$, where each literal x_{ji}^{pi} is either a propositional variable $x_{ji}^1 = x$ or its negation $x_{ji}^0 = \bar{x}$. All remaining lines are passed

Just Accepted

unaltered. Note that whenever t is equal to 0, the MPMC Toffoli gate computes the AND of all control lines on the target line.

For example, an MPMC Toffoli gate with positive control x_1, negative control x_2, and target line t maps

$$t \mapsto t \oplus (x_1 \wedge \bar{x}_2).$$

Thus, if the target line is initialized to 0, the gate computes the product term $x_1\bar{x}_2$ on the target line. This illustrates the correspondence between product terms in ESOP expressions and MPMC Toffoli gates: each product term can be implemented by one gate whose controls are the literals of the product and whose target is the output line.

A common approach in reversible logic design is ESOP-based synthesis, which is based on the relationship between product terms in ESOP (Exclusive Sum of Products) forms and MPMC Toffoli gates [14, 15, 18]. In an ESOP expression, MPMC Toffoli gates are derived from product terms, with their literals serving as control lines. The circuit effectively computes the XOR of all product terms on the same target line because all gates act on the same target line.

As a preliminary step towards building quantum circuits, reversible circuit synthesis plays an important role in quantum computing. A quantum compilation step is then required to convert a reversible circuit containing MPMC Toffoli gates into a quantum circuit. In this process, each reversible gate can be decomposed into basic quantum gates, following a specified quantum gate library [24, 27]. In this work, we use the Clifford+T library, which includes Pauli, Hadamard, CNOT gates, and the T-gate, which is considered the most expensive [32].

Recent advances in quantum compilation techniques have led to more efficient circuit designs. In [26], for instance, a novel method is introduced and analyzed for compiling quantum circuit representations from XAG (XOR-And Graph) representations, reducing the number of T-gates required to implement Toffoli gates.

Finally, we refer the reader to [37] and [46] for more detailed information on reversible circuits and effective quantum compilation strategies.

3 Quantum Circuit Synthesis for autosymmetric and D-reducible functions

As discussed in the previous section, there is a natural correspondence between ESOP expressions and reversible circuits. This arises from the fact that product terms in an ESOP expression can be effectively represented using reversible MPMC Toffoli gates. Given an ESOP expression, one can generate a sequence of MPMC Toffoli gates, where the control lines are associated with the literals in the product terms, and the target line represents the output of the function. Note that all gates act on the same target line, thereby implementing the exclusive OR of the product terms.

The primary challenge with this approach lies in the fact that MPMC Toffoli gates are only intermediate representations. To implement these gates in quantum circuits, an additional synthesis step is required to map the reversible MPMC Toffoli gates into functionally equivalent quantum circuits. Unfortunately, this mapping step can become very onerous, particularly when the MPMC Toffoli gates are controlled by many variables, leading to quantum circuits with significant size and depth.

To address these challenges, in this section, we propose a comprehensive approach for designing circuits that implement functions exhibiting both autosymmetry and D-reducibility. These methods are intended to optimize the reversible synthesis process, leading to quantum circuits that are efficient and minimized in terms of size and depth. The methodology is divided into two distinct strategies, depending on the order in which autosymmetry and D-reducibility are considered during circuit construction.

3.1 Autosymmetric-D-Reducible Design

The first approach focuses on functions that are first analyzed for autosymmetry and then subjected to D-reducibility. Initially, an autosymmetry test is performed to determine the linear space L and compute the restricted function f_k, which corresponds to the autosymmetric structure of the original function. Once the

Just Accepted

autosymmetry analysis is complete, D-reducibility is applied to the restricted function f_k, in order to identify the characteristic function χ_A of the affine space A that contains the minterms of f_k. The restricted function f_k is then projected onto the affine space A, resulting in the simplified function f_{kA}. This sequential decomposition approach highlights the interplay between autosymmetric properties and affine projections, where autosymmetry is applied first to reduce the complexity of the function, and D-reducibility is subsequently used to further simplify it by projecting onto an affine space. The circuit design under this approach involves the integration of three key reversible circuits, followed by an additional uncomputing stage:

- Circuit for Deriving Reduction Equations: The initial circuit is designed to derive the reduction equations for the linear space L. These equations involve both canonical and non-canonical variables within L. The primary aim of this circuit is to simplify the function by reducing the number of variables, thus preparing the function for subsequent processing.
- Circuit for the Characteristic Function χ_A: The next circuit computes the characteristic function χ_A of the affine space A covering f_k. This function depends on non-canonical variables within the linear space L, which includes both canonical and non-canonical variables within the affine space A. Computing χ_A is crucial as it provides the necessary foundation for the final projection step.
- Circuit for Implementing Function Projection f_{kA}: The next circuit implements the projection of the restricted function f_k onto A, resulting in f_{kA}. This projection circuit focuses on the non-canonical variables within the affine space A, ensuring that the function is further simplified and optimized for the final quantum synthesis.
- Circuit for Restoring Initial Variables: This stage in the synthesis process implements the so-called *uncomputing* procedure, i.e., it restores the initial values of the canonical and non-canonical variables of the linear space L after performing the required computations.

To compute the final function f, we use a Toffoli gate to perform an AND operation between the subfunctions χ_A and f_{kA}, with f as the output target. This structured approach ensures that the synthesis of the quantum circuit is both streamlined and efficient, minimizing resource usage while maintaining the integrity of the function.

Figure 6 illustrates the overall structure of the circuit designed using this method. As shown, this approach requires three additional lines, corresponding to three extra qubits in the quantum implementation: one for χ_A, one for f_{kA}, and one for f. These additional qubits are essential for handling the intermediate results of the subfunctions before combining them to produce the final output.

Overall, the implementation of the circuit begins with the construction of the XOR layer. This layer, composed of multiple XOR gates, is necessary to realize the reduction equations, which are based on linear combinations of the variables. The XOR layer is a critical component, as it enables the reduction process by simplifying the function's dependency on certain variables, making the final quantum circuit more compact and efficient. Then, a reversible circuit for χ_A can be implemented directly from the CEX expression of the affine space A, and a reversible circuit for f_{kA} depends only on the canonical variables of the affine space A. Then, an uncomputing procedure restores all variables to their initial values. Finally, an MPMC Toffoli gate is needed to implement the AND of χ_A and f_{kA}. The overall methodology is summarized in Algorithm 1.

Observe that the circuit shown in Figure 6, and derived by applying Algorithm 1, illustrates only the computation stage of the proposed synthesis strategy. During the computation, any input line whose value is temporarily modified is restored to its original value by the end of the circuit, while the intermediate values $\chi_A(x)$ and f_{kA} are left on the two additional output lines to explicitly illustrate the decomposition process. Since these values are not required for any subsequent computation, the corresponding output lines may be regarded as garbage outputs and are therefore not uncomputed in the presented circuits. When a clean quantum oracle is required, these intermediate values can be removed by applying the standard uncomputation procedure after the target

Just Accepted

function f has been generated. This restores the two auxiliary output lines to their initial values while preserving the value computed for f.

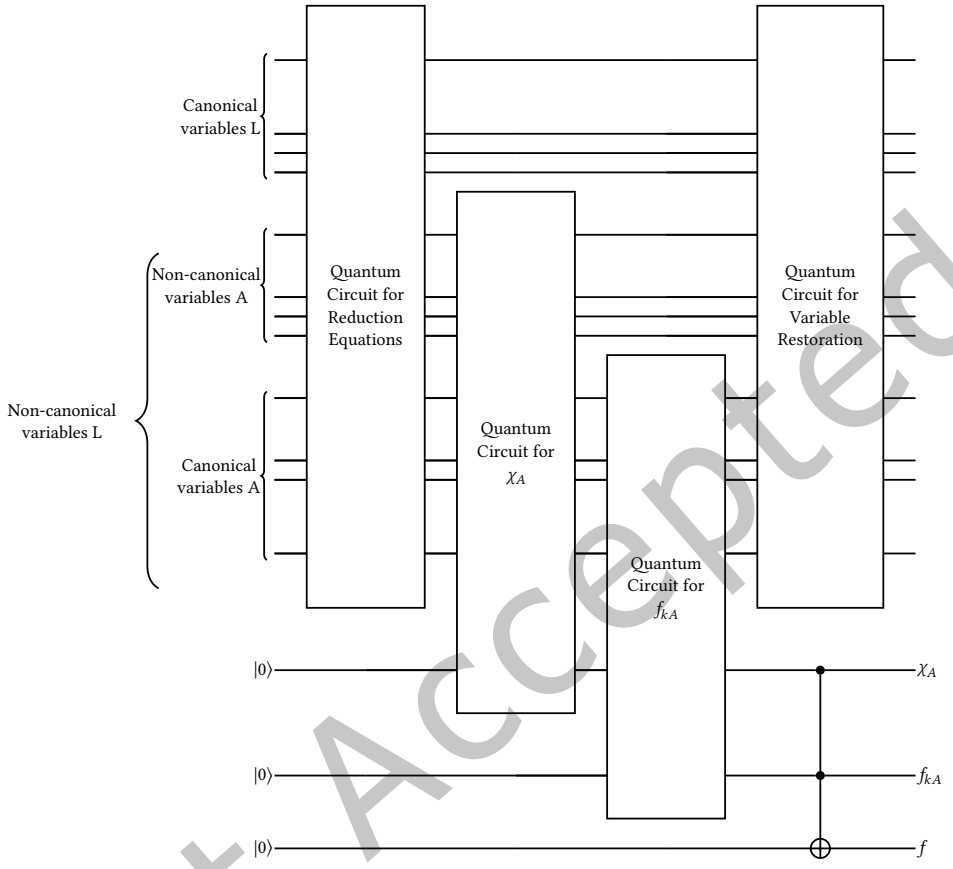


Fig. 6. Reversible circuit with the Autosymmetric–D-Reducible strategy. For clarity, only the computation stage is shown. The uncomputation stage is omitted. A clean quantum oracle can be obtained by applying the inverse computation after the target output has been generated.

Just Accepted

Algorithm 1 Reversible synthesis of autosymmetric–D-reducible functions

Require: Boolean function f that is both autosymmetric and D-reducible
Ensure: Reversible circuit Q implementing f

```
(L_f, E_L, f_k) ← AutosymmetryDecomposition(f)      ▷ E_L contains the reduction equations induced by L_f
(A, χ_A, f_kA) ← DReducibilityDecomposition(f_k)    ▷ χ_A is the characteristic function of the affine space A
Q ← empty reversible circuit
Q_E ← InPlaceXORNetwork(E_L)                        ▷ realize the reduction equations with CNOT gates
Append Q_E to Q
Q_χ ← ReversibleSynthesis(χ_A)                      ▷ compute the characteristic function on an auxiliary line
Append Q_χ to Q
Q_kA ← ReversibleSynthesis(f_kA)                    ▷ compute the projected function on an auxiliary line
Append Q_kA to Q
Append a Toffoli gate controlled by χ_A and f_kA to update the output line of f
Append Q_E^{-1} to Q                                ▷ restore the variables modified by the XOR network
return Q
```

Example 9. *Consider the function*

$$f = \{00011, 01010, 01110, 10001, 10101, 11000\}$$

in the running example. To compute the reduction equations defining the new variables y_1, y_2, y_3, y_4, we use three CNOT gates to perform the XOR operations $x_1 \oplus x_2$, $x_1 \oplus x_4$ and $x_1 \oplus x_5$. As mentioned in Section 2.1, a CEX consists of a conjunction of XOR factors, with each XOR factor being implementable using CNOT gates. In particular, an XOR factor with k literals can be realized using $k-1$ CNOT gates. In this case, for χ_A we use a CNOT gate to perform $y_1 \oplus y_4$, and a Toffoli gate that combines y_3 with the result of $y_1 \oplus y_4$. The projection f_{kA} can be computed by a Toffoli gate to perform $y_1 + \overline{y}_2$ operations. Recall that, to simplify the description of our example, we represent the function f_{kA} in SOP form. Note that f_{kA} can be expressed in various forms, but we will use the XAG representation in the experimental section. Subsequently, the next Toffoli gate computes the AND between χ_A and f_{kA}. Finally, four additional gates are used to restore the input variables. As can be observed in Figure 7, this quantum implementation requires three Toffoli and eight CNOT gates.

In this example, four T-gates suffice to represent χ_A, as proved in the following proposition.

Proposition 1. *Let $\chi_A : \{0,1\}^n \to \{0,1\}$ be the canonical CEX expression representing an affine subspace A of $\{0,1\}^n$, where the number of factors in χ_A is denoted by $|\mathcal{F}|$. Then, a quantum circuit for χ_A can be implemented using $4 \cdot (|\mathcal{F}| - 1)$ T-gates.*

Proof. The function $\chi_A : \{0,1\}^n \to \{0,1\}$ represents the characteristic function of an affine subspace $A \subseteq \{0,1\}^n$, thus it is a conjunction (AND) of XOR factors. If the number of factors in the expression for χ_A is $|\mathcal{F}|$, then the number of AND gates required to implement χ_A is $|\mathcal{F}| - 1$. This is because each AND gate operates on two inputs, and to compute a conjunction of $|\mathcal{F}|$ terms, we need $|\mathcal{F}| - 1$ binary AND operations.

Each AND gate in the circuit is implemented by a Toffoli gate. In our cost analysis, we assume the ancilla-assisted implementation proposed by Gidney [16], in which an exact Toffoli gate is realized using four T gates. Consequently, each AND operation contributes four T gates to the overall T-count, while CNOT gates do not contribute any T gates. Therefore, the total number of T gates required to implement χ_A is $4 \cdot (|\mathcal{F}| - 1)$. It should be noted that the four-T implementation achieves this reduction by trading T gates for additional quantum resources. Specifically, it requires one clean ancillary qubit together with measurement and classical feed-forward.

Just Accepted

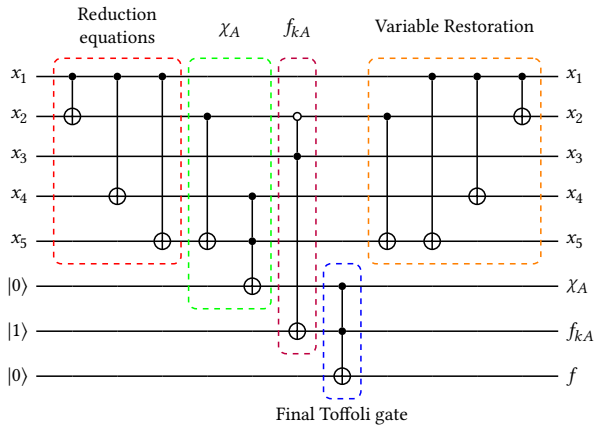


Fig. 7. Reversible circuit for the function f with Autosymmetric-D-Reducible strategy.

By contrast, the standard ancilla-free decomposition of the Toffoli gate requires seven T gates. Thus, the T-count reported in this proposition is obtained under the assumption of the ancilla-assisted implementation. □

Moreover, we have that

Proposition 2. *The XOR network induced by the autosymmetry reduction equations can be implemented in-place using only CNOT gates and without additional ancilla lines.*

Proof. In the canonical representation of the autosymmetry space, every reduction equation is associated with one distinct non-canonical variable. Let x_j be such a non-canonical variable. Its reduction equation has the form

$$y_j = x_j \oplus (\bigoplus_{x_i \in C_j} x_i),$$

where C_j is a subset of canonical variables. This equation can be realized by applying CNOT gates controlled by the variables in C_j and targeted on the line x_j. Thus, the line x_j is overwritten with y_j, while the canonical variables are used only as controls and are not modified. Since distinct reduction equations have distinct non-canonical variables as their targets, the in-place implementation introduces no target conflicts. Moreover, the whole XOR layer is a reversible linear transformation, and its inverse is obtained by applying the same CNOT gates in reverse order. Therefore, the original variables can be restored during the uncomputation stage, and no additional ancilla line is needed for implementing the XOR layer. □

3.2 D-Reducible–Autosymmetric Design

The second approach reverses the order of the analysis, starting with the D-reducibility before addressing the autosymmetry. In this strategy, the process begins with the identification of the characteristic function χ_A and also the projection f_A, which are derived from the D-reducibility test. Once these components are established, the autosymmetry properties of the function are considered and tested directly onto f_A, leading to the reduction equations and the restriction function f_{Ak}. Hence, the final representation has the form

$$f = \chi_A \cdot f_{Ak}.$$

Just Accepted

Algorithm 2 Reversible synthesis of D-reducible–autosymmetric functions

Require: Boolean function f that is both D-reducible and autosymmetric
Ensure: Reversible circuit Q implementing f

```
(A, χ_A, f_A) ← DReducibilityDecomposition(f)   ▷ χ_A is the characteristic function of the affine space A
(L_{f_A}, E_L, f_Ak) ← AutosymmetryDecomposition(f_A) ▷ E_L contains the reduction equations induced by L_{f_A}
Q ← empty reversible circuit
Q_χ ← ReversibleSynthesis(χ_A)                  ▷ compute the characteristic function on an auxiliary line
Append Q_χ to Q
Q_E ← InPlaceXORNetwork(E_L)                    ▷ realize the reduction equations with CNOT gates
Append Q_E to Q
Q_Ak ← ReversibleSynthesis(f_Ak)                ▷ compute the restricted projected function on an auxiliary line
Append Q_Ak to Q
Append a Toffoli gate controlled by χ_A and f_Ak to update the output line of f
Append Q_E^{-1} to Q                            ▷ restore the variables modified by the XOR network
return Q
```

At the circuit level, the order of the blocks follows this decomposition. The circuit first computes χ_A on an auxiliary line, using the CEX expression of the affine space A. Then, an in-place XOR network realizes the autosymmetry reduction equations for f_A. After this linear transformation, a reversible subcircuit computes the restricted function f_{Ak} on a second auxiliary line. Finally, a Toffoli gate controlled by χ_A and f_{Ak} updates the oracle target line f, and the inverse XOR network restores the variables modified by the reduction equations.

Thus, the main difference with respect to the A+D construction is that χ_A is computed before the autosymmetry XOR layer, whereas in the A+D construction the XOR layer is applied before computing the characteristic function of the affine space associated with the restricted function. As discussed in Section 3.3, this different ordering may lead to different CNOT counts and different intermediate XAG representations, even when the two decompositions are functionally equivalent.

Figure 8 shows the reversible circuit obtained with this strategy, and the corresponding synthesis procedure is summarized in Algorithm 2. As before, the circuit illustrates only the computation stage of the proposed synthesis strategy. Since the intermediate values $\chi_A(x)$ and f_{Ak} are not required for any subsequent computation, the corresponding output lines may be regarded as garbage outputs and are therefore not uncomputed in the presented circuits. When a clean quantum oracle is required, these intermediate values can be removed by applying the standard uncomputation procedure after the target function f has been generated. This restores the two auxiliary output lines to their initial values while preserving the value computed for f.

Example 10. *Consider the function from the running example once again. The detailed process for deriving χ_A is provided in Section 2.1. We then use two CNOT gates to perform $x_1 \oplus x_4$ and $x_1 \oplus x_5$, and lastly a Toffoli gate combining the result of $x_1 \oplus x_4$ with the result of $x_1 \oplus x_5$. We also use a CNOT gate to perform the reduction equations y_1 and y_2. To compute f_{Ak}, a Toffoli gate is used to compute the $y_1 + \overline{y_2}$ operation. Then, the next Toffoli gate computes the AND between χ_A and f_{Ak}. Finally, three additional gates are used to restore the original input variables. As depicted in Figure 9, this quantum implementation requires three Toffoli gates and six CNOT gates. It should be noted that, according to Proposition 1, the number of T-gates for χ_A in this example is equal to four.*

We finally prove that a reversible circuit for the target function f obtained with the Autosymmetric-D-Reducible and D-Reducible-Autosymmetric strategies can be implemented with just three additional output lines.

Just Accepted

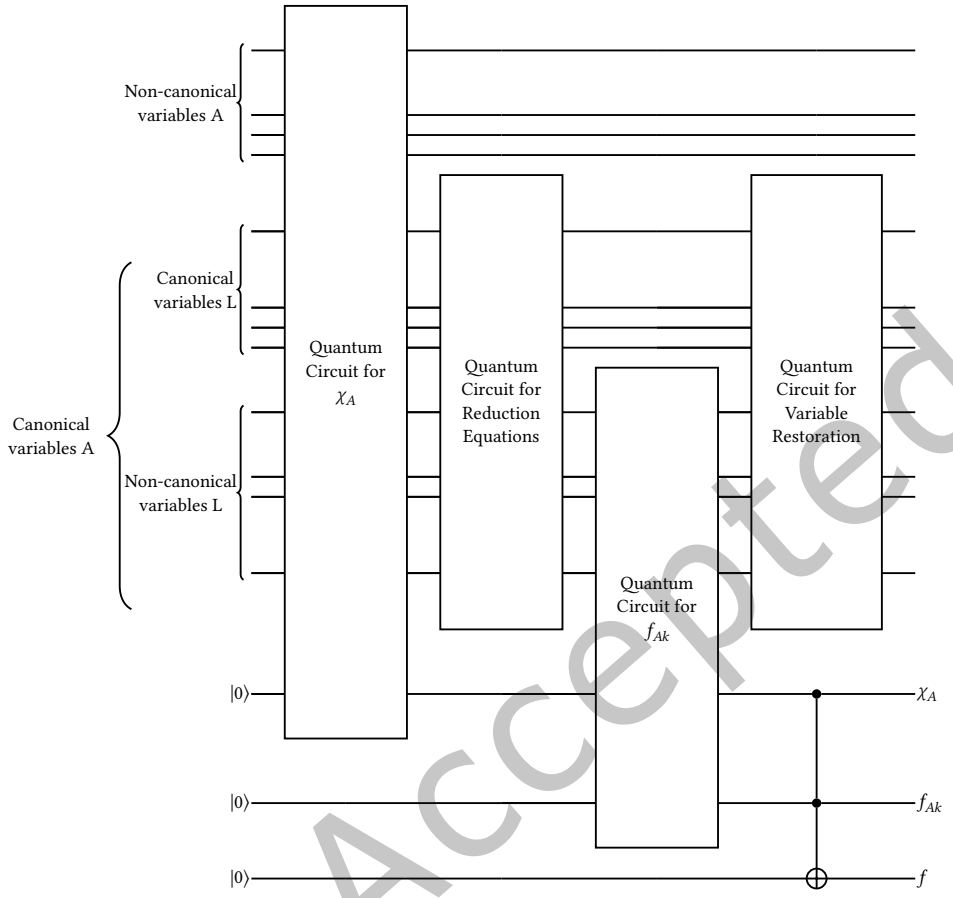


Fig. 8. Reversible circuit with D-Reducible-Autosymmetric strategy. For clarity, only the computation stage is shown. The uncomputation stage is omitted. A clean quantum oracle can be obtained by applying the inverse computation after the target output has been generated.

Just Accepted

Proposition 3. *Let $\chi_A : \{0,1\}^n \to \{0,1\}$ be the canonical CEX expression representing an affine subspace A of $\{0,1\}^n$, and let f_{kA} (or equivalently f_{Ak}) be the function projected onto the affine subspace A. Reversible circuits for f, constructed using either the Autosymmetric-D-Reducible or D-Reducible-Autosymmetric strategy, require only three additional output lines: one line for the target function f, and two lines for storing the intermediate results χ_A and f_{kA} (or f_{Ak}). The implementation of each Toffoli gate in these circuits may also require ancillary qubits, with the number of ancillas depending on the number of control qubits used in the gates.*

Proof. The implementation of χ_A involves logical AND operations, which are realized using Toffoli gates or, when there are more than two factors, multi-controlled Toffoli gates. The implementation of multi-controlled Toffoli gates may require additional *logical* ancillary qubits, whose number depends on both the number of control qubits and the adopted gate decomposition. Similarly, the function f_{kA} (or f_{Ak}) is computed using Toffoli and multi-controlled Toffoli gates with similar logical ancillary requirements. In addition, under the Clifford+T cost model adopted in this paper, each Toffoli gate is implemented using the ancilla-assisted four-T decomposition

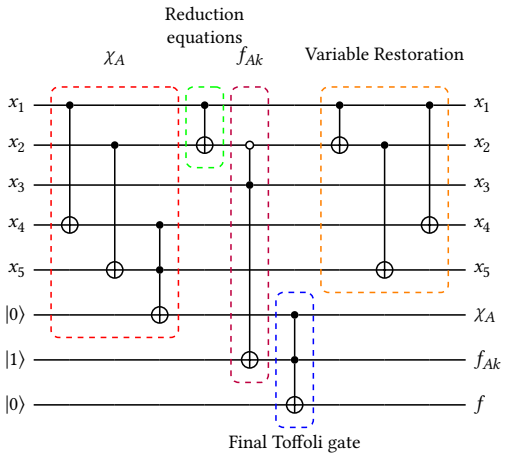


Fig. 9. Reversible circuit for the function f with D-Reducible-Autosymmetric strategy.

proposed by Gidney [16]. Consequently, each Toffoli gate requires one additional clean ancillary qubit together with measurement and classical feed-forward. These ancillas are distinct from the logical ancillas required for the implementation of multi-controlled Toffoli gates.

The proposed circuit introduces only two additional output lines, in addition to the output line reserved for the target function f: one for storing χ_A and one for storing f_{kA} (or f_{Ak}), ensuring that these intermediate results are preserved for subsequent computation and for the final reconstruction of f. Therefore, independently of the ancillary qubits required by the underlying gate implementations, the proposed synthesis strategy requires a total of three additional output lines, as shown in Figures 6 and 8.

□

Proposition 2 holds also in this case. Overall, both approaches provide a systematic and efficient pathway for designing quantum circuits that handle functions with both autosymmetry and D-reducibility, ensuring that the final circuits are not only functional but also optimized for quantum implementation.

3.3 Impact of the Decomposition Order on Circuit Construction

For completely specified Boolean functions that are both autosymmetric and D-reducible, the two decompositions A+D and D+A lead to the same final reduced Boolean function, as recalled in Section 2.3. However, the equality $f_{Ak} = f_{kA}$ is a functional equality and does not imply that the two reversible circuit constructions are identical. The reason is that the two strategies compute the same final function through different intermediate representations.

In the A+D strategy, the autosymmetry reduction is applied first. Therefore, the first part of the circuit is an implementation of the reduction equations induced by the autosymmetry space L_f. The characteristic function χ_A is then computed in the reduced space, and the projected function f_{kA} is synthesized from this reduced representation.

In the D+A strategy, the order is different. The characteristic function χ_A of the affine space associated with the original function is computed before the autosymmetry reduction equations are applied. The XOR network is then used only for the projected function f_A, and the final reduced function f_{Ak} is synthesized after this projection. Thus, even when $f_{Ak} = f_{kA}$, the two circuits contain different sequences of CNOT gates and different intermediate signals.

Just Accepted

Example 11. *The running example illustrates this effect. In the A+D construction, the circuit first realizes the reduction equations*

$$y_1 = x_1 \oplus x_2, \qquad y_2 = x_3, \qquad y_3 = x_1 \oplus x_4, \qquad y_4 = x_1 \oplus x_5,$$

which requires three CNOT gates before computing χ_A and f_{kA}. The same CNOT network must then be applied in reverse during the uncomputing stage to restore the initial values of the non-canonical variables, modified by the implementation of the reduction equations. In the D+A construction, instead, χ_A is computed directly from the affine-space representation of the original function before applying the autosymmetry reduction to f_A. Consequently, fewer CNOT gates are needed in the XOR network of this example. As shown in Figures 7 and 9, the A+D circuit requires three Toffoli gates and eight CNOT gates, whereas the D+A circuit requires three Toffoli gates and six CNOT gates.

The difference also affects XAG-based quantum compilation such as [45]. The XAG heuristic is applied to the intermediate function produced by the chosen decomposition order. Although the final Boolean function is equivalent, the intermediate XAGs may have different numbers of XOR and AND nodes, different sharing opportunities, and different local structures. Since the XAG synthesis and optimization procedure is heuristic, semantically equivalent expressions may be compiled into circuits with different T-counts and CNOT-counts. This explains why the two decomposition orders may lead to different quantum costs even for completely specified functions.

3.4 Preprocessing step and circuit-generation complexity

The proposed synthesis flows consist of a classical preprocessing phase followed by XAG-based quantum compilation. The preprocessing phase is not part of the quantum circuit: it is performed to detect the regularities of the input function and to construct the reduced functions used for synthesis.

The D-reducibility test computes the affine space A associated with the onset of the function. This step is based on linear-algebra operations over $\mathbb{F}_2$, in particular on the computation of a basis of the corresponding vector space, and can be carried out in polynomial time and polynomial space with respect to the size of the input [6]. Once the canonical basis and the translation point of A are known, the CEX expression of the characteristic function χ_A is obtained by deriving one XOR factor for each non-canonical variable. Hence, the construction of χ_A is time and space polynomial in the number of variables and in the size of the basis.

Similarly, the autosymmetry test computes the autosymmetry space L_f, the reduction equations, and the restricted function f_k. When the function is represented by an ROBDD [7], the complexity of this computation is polynomial in time and space with respect to the size of that representation. The reduction equations are then translated into an in-place CNOT network whose size is linear in the total number of variables appearing in the equations.

In the A+D strategy, the autosymmetry test is first applied to the input function f, producing L_f, the reduction equations, and f_k. Then, the D-reducibility test is applied to f_k, producing the affine space A, the characteristic function χ_A, and the projected function f_{kA}. The projected function f_{kA} is obtained by rewriting f_k in the coordinates induced by the affine space A. In the D+A strategy, the order is reversed: the D-reducibility test is first applied to f, producing χ_A and f_A, and the autosymmetry test is then applied to f_A, producing the reduction equations and the restricted function f_{Ak}. Therefore, in both strategies, the preprocessing phase requires polynomial time and polynomial space with respect to the adopted input representation and the number of variables.

After the regularity tests, the reduced functions f_{kA} or f_{Ak} are passed to the XAG-based synthesis and quantum compilation backend (e.g., [45]). This optimization step is heuristic and implementation-dependent; therefore, we do not claim worst-case optimality for the produced XAGs. However, once an XAG representation is computed,

Just Accepted

the generation of the corresponding reversible circuit is linear in the number of XAG nodes, with one local circuit block for each XOR or AND node.

4 Experimental results

In this section, we assess the effectiveness of the proposed methods for quantum synthesis of autosymmetric and dimension reducible functions. We present the computational results obtained by constructing decomposed expressions for Boolean functions and comparing them to their standard quantum synthesis counterparts, which do not exploit these structural regularities. This experiment aims to measure the extent to which our strategies can be exploited to implement more compact circuits. In the following, the term *standard quantum synthesis* denotes the direct synthesis of the original single-output Boolean function f, without applying either autosymmetry or D-reducibility as a preprocessing decomposition step. More precisely, the baseline is obtained by applying the XAG-based quantum compilation flow directly to f. We compare this baseline with two preprocessing-based flows. In the A+D flow, we first apply the autosymmetry test to f, then the D-reducibility test to the restriction f_k, and finally apply XAG-based quantum compilation to the projected function f_{kA}. In the D+A flow, we first apply the D-reducibility test to f, then the autosymmetry test to the projection f_A, and finally apply XAG-based quantum compilation to the restricted function f_{Ak}. In all cases, the same XAG-based quantum compilation backend is used; the only difference among the three flows is whether and how the structural preprocessing is applied before compilation.

All experiments were conducted on an Intel i7-8550U CPU running at 1.80GHz with 16GB of RAM. We used classical benchmarks in PLA format, specifically from the Espresso and LGSynth'89 benchmark suites [47]. It should be noted that autosymmetry and D-reducibility are properties of single outputs, which means that different outputs from the same benchmark can have different autosymmetry degrees and D-reducibility properties. As a result, each output is considered as a separate Boolean function for analysis, leading to the examination of 218 autosymmetric and D-reducible functions. The given functions, along with their restrictions or projections, were decomposed using the structural strategies depicted in Figures 6 and 8, with ESPRESSO [11] employed in heuristic mode for SOP synthesis, and then they were synthesized by applying the XAG-based quantum compilation heuristic proposed in [45].

Our primary focus was to experimentally determine whether advanced quantum compilation methods could benefit from the decomposition based on D-reducibility and autosymmetry or if they are capable of deriving compact quantum circuits, even without exploiting these structural regularities.

To evaluate the quantum circuits obtained after Clifford+T compilation from the proposed decomposition strategies and compare them with those obtained without decomposition, we measured the cost in terms of elementary quantum T-gates and CNOT gates. Specifically, we map each MPMC Toffoli gate into elementary quantum gates based on the Clifford+T library and the algorithm described in [26]. Since the T-gate is recognized as the most expensive gate in the library, the cost of a Toffoli gate is typically expressed in terms of the number of T-gates required for its implementation. Therefore, in this experimental evaluation, we consider the number of T-gates in the circuits obtained by applying the XAG-based quantum compilation heuristic with and without exploiting the decomposition based on the D-reducibility-autosymmetry and autosymmetry-D-reducibility strategies. In addition, we also report the CNOT-count, since CNOT gates typically dominate the two-qubit gate cost and can significantly impact circuit depth and fidelity on realistic hardware [2, 25]. Hence, for each benchmark we provide both T-count and CNOT-count for the baseline compilation flow and for the two decomposition orders.

As already mentioned, a notable result highlighted in [26] indicates that the number of T-gates in a quantum circuit for a Boolean function f can be expressed as a function of the number of AND gates in its XAG representation. Moreover, the CNOT-count is influenced by the number of XOR operations and by the (un)computation

Just Accepted

Table 1. Representative comparison between standard synthesis and the proposed strategies for autosymmetric and D-reducible functions from the Espresso and LGSynth'89 benchmark sets.

	Standard Synthesis		With Decomposition				T-Gain		CNOT-Gain	
Benchmark	**T-count**	**CNOT**	**T-count (A+D)**	**CNOT (A+D)**	**T-count (D+A)**	**CNOT (D+A)**	**A+D**	**D+A**	**A+D**	**D+A**
bcd.div3_2	12	40	12	3	**8**	3	0%	33%	93%	93%
bench_3	32	8	**12**	12	**12**	12	63%	63%	-50%	-50%
bench_4	24	6	**16**	11	**16**	7	33%	33%	-83%	-17%
cps_0	172	96	**160**	74	**160**	74	7%	7%	23%	23%
dk17_0	28	8	**16**	11	**16**	3	43%	43%	-38%	63%
dk17_1	28	8	**16**	11	**16**	3	43%	43%	-38%	63%
dk17_10	24	30	**12**	11	**12**	3	50%	50%	63%	90%
dk17_4	40	14	**16**	17	20	14	60%	50%	-21%	0%
dk27_2	20	8	**12**	16	**12**	4	40%	40%	-100%	50%
dk48_4	52	8	20	38	**8**	5	62%	85%	-375%	38%
exam_7	60	12	**24**	13	**24**	17	60%	60%	-8%	-42%
exp_11	60	36	24	25	**20**	10	60%	67%	31%	72%
exps_18	36	6	52	3	**28**	3	-44%	22%	50%	50%
exps_19	48	6	52	6	**28**	6	-8%	42%	0%	0%
exps_31	28	30	36	5	**20**	4	-29%	29%	83%	87%
p1_10	48	34	20	28	**16**	21	58%	67%	18%	38%
p1_11	44	46	**16**	12	**16**	9	64%	64%	74%	80%
rd84_1	12	48	**4**	7	**4**	7	67%	67%	85%	85%
t1_19	20	8	20	3	**12**	3	0%	40%	63%	63%
xor5_0	12	6	**4**	4	**4**	4	67%	67%	33%	33%

overhead introduced by the mapping, making it a complementary indicator to assess the impact of the proposed decompositions on the overall gate complexity.

Table 1 reports a significant subset of benchmarks as representative indicators of our experiments. The first column reports the name and number of the considered autosymmetric and D-reducible output of each benchmark. The second column reports the costs, in terms of T-count and CNOT-count, of the quantum circuits obtained after Clifford+T compilation of the XAG expressions of the benchmarks without exploiting structural regularities. The following columns report the costs, in terms of T-count and CNOT-count, of the quantum circuits obtained after Clifford+T compilation of the decomposed implementation produced by the autosymmetry-D-reducibility strategy, as explained in Section 3.1. The next columns report the corresponding costs for the quantum circuits obtained from the D-reducibility-autosymmetry strategy, as described in Section 3.2. The last two groups of two columns report the T-gain and CNOT-gain of the compiled quantum circuits obtained from the autosymmetry-D-reducibility and D-reducibility-autosymmetry strategies, respectively. SOP minimization of the benchmarks and their projections onto the associated affine spaces is performed using the Espresso heuristic [11]. Due to the heuristic nature of this SOP minimizer, the synthesis times for the functions and their projections are similar and very short, leading to negligible gains in synthesis time. As shown in Table 1, some benchmarks experience significant advantages from the decomposed implementations in terms of T-count and CNOT-count compared to standard synthesis. For instance, for the benchmarks *exp_11* and *rd84_1*, we significantly reduce the number of T-gates when utilizing D-reducibility-autosymmetry and autosymmetry-D-reducibility decomposition, respectively. Specifically, *exp_11* shows a 67% reduction in T-gates and 72% in CNOTs, while *rd84_1* shows a 67% reduction in T-gates and 85% in CNOTs. However, in some cases, standard synthesis results in circuits with fewer T-gates or CNOTs compared to synthesis that exploits structural regularities, as seen, for example, in the *bench_3* or *exps_18* benchmarks. This is due to the heuristic nature of the proposed method.

To provide a complete view of the experimental results, we also report the number of improved, unchanged, and worsened instances over the full set of 218 autosymmetric and D-reducible functions. For T-count, the

Just Accepted

A+D strategy improves 69 instances, leaves 21 unchanged, and worsens 128, while the D+A strategy improves 90 instances, leaves 30 unchanged, and worsens 98. For CNOT-count, A+D improves 69 instances, leaves 75 unchanged, and worsens 74, while D+A improves 78 instances, leaves 79 unchanged, and worsens 61.

Table 2 reports aggregate statistics restricted to the non-negative-gain instances, i.e., the cases in which the decomposed circuit has a cost lower than or equal to that of the standard synthesis flow. The choice between the decomposed implementations and the standard one is made with respect to the selected cost metric. In a fault-tolerant setting, this metric is typically the T-count, whereas in near-term or hardware-constrained settings, the CNOT-count may be more relevant. For each benchmark output, we synthesize the standard, A+D, and D+A implementations and retain the one with the lowest value of the preselected metric. This metric-dependent behavior is already visible in Table 1. For instance, for bench_3, both decomposition orders reduce the T-count by 63%, but increase the CNOT-count by 50%. Conversely, for exps_18 with the A+D order, the CNOT-count improves by 50%, while the T-count worsens by 44%. Percentage-based statistics and non-negative gain counts are computed only for instances with a non-zero baseline cost. Since unchanged instances are included in the non-negative-gain subset, the worst non-negative gain is always 0%; therefore, the table reports only the best case and does not include a separate worst-case column.

Table 2. Aggregate statistics over the non-negative-gain instances from the Espresso and LGSynth'89 benchmark sets.

Metric	Strategy	Non-negative cases	Mean gain	Median gain	Aggregate gain	Best case
T-count	A+D	90	31%	23%	27%	rd73_1 83%
T-count	D+A	120	29%	22%	27%	dk48_4 85%
CNOT	A+D	76	49%	50%	44%	bcd.div3_2 93%
CNOT	D+A	87	54%	53%	54%	risc_4 100%

On this subset, the aggregate T-count reductions are 27% for A+D and 27% for D+A. For CNOT-count, the corresponding aggregate reductions are 44% for A+D and 54% for D+A. Thus, the reductions reported in the introduction and in the discussion of Table 1 refer to this cost-driven non-negative-gain setting, while the full-set behavior is summarized by the improved, unchanged, and worsened counts reported above.

It should be mentioned that running times are significantly influenced by the XAG heuristic [45], which generally varies according to the dimension of the input function. Consequently, when both testing procedures are performed, the overall running times are often reduced, as the input function for the XAG heuristic tends to be smaller. In other words, the time saved in constructing the XAG outweighs the time needed to test the two regularities. Furthermore, for completely specified functions, where the results of A+D and D+A are the same, the D+A strategy is more convenient due to its superior running times [3].

Overall, on the Espresso and LGSynth'89 benchmark set, we can conclude that the compiled quantum circuits obtained from the decomposition scheme with D-reducibility first, followed by autosymmetry, tend to outperform those obtained from the scheme where autosymmetry is applied before D-reducibility, under the considered T-count and CNOT-count metrics.

To further evaluate the effect of the proposed decompositions, we also considered the EPFL benchmark set [1]. This experiment includes 395 single-output functions. For each function, we compare the standard synthesis flow with the two decomposition orders, namely autosymmetry followed by D-reducibility (A+D) and D-reducibility followed by autosymmetry (D+A).

Table 3 reports a representative subset of the EPFL results. The table includes cases with large improvements, cases where both T-count and CNOT-count improve, and cases where the decomposition reduces the T-count

Just Accepted

Table 3. Representative comparison between standard synthesis and the proposed strategies for autosymmetric and D-reducible functions from the EPFL benchmark set.

	Standard Synthesis		With Decomposition				T-Gain		CNOT-Gain	
Benchmark	**T-count**	**CNOT**	**T-count (A+D)**	**CNOT (A+D)**	**T-count (D+A)**	**CNOT (D+A)**	**A+D**	**D+A**	**A+D**	**D+A**
max_po127	6100	5728	**12**	0	**12**	0	99%	99%	100%	100%
div_po17	31760	12712	**26556**	193472	**26556**	193472	16%	16%	-1422%	-1422%
div_po20	28212	11120	**23356**	158256	23360	158256	17%	17%	-1323%	-1323%
div_po28	19480	7384	**15772**	85304	**15772**	85304	19%	19%	-1055%	-1055%
div_po37	11616	4096	**9028**	35172	9032	35172	22%	22%	-759%	-759%
div_po59	1208	182	**696**	206	700	206	42%	42%	-13%	-13%
ctrl_po0	32	18	**16**	34	**16**	34	50%	50%	-89%	-89%
ctrl_po2	12	58	**20**	4	**20**	4	-67%	-67%	93%	93%
ctrl_po11	20	20	**16**	0	**16**	0	20%	20%	100%	100%
ctrl_po20	52	18	**20**	40	**20**	40	62%	62%	-122%	-122%
cavlc_po4	40	4	**24**	8	**24**	8	40%	40%	-100%	-100%
i2c_po17	132	8	**112**	0	**112**	0	15%	15%	100%	100%
i2c_po35	128	8	**108**	8	**108**	8	16%	16%	0%	0%
i2c_po44	88	96	**112**	60	**112**	60	-27%	-27%	38%	38%
mem_ctrl_po238	2712	2572	**1508**	1004	1720	1680	44%	37%	61%	35%
mem_ctrl_po256	388	6	**264**	4	280	0	32%	28%	33%	100%
mem_ctrl_po254	420	8	**400**	54	**400**	54	5%	5%	-575%	-575%
mem_ctrl_po271	1248	142	**1004**	70	1012	62	20%	19%	51%	56%
mem_ctrl_po724	500	250	**352**	200	472	164	30%	6%	20%	34%
mem_ctrl_po201	1868	240	**1488**	140	1496	132	20%	20%	42%	45%

but increases the CNOT-count. This trade-off is particularly visible on some outputs of the `div` benchmark. For example, for `div_po17`, both decomposition orders reduce the T-count by 16%, but increase the CNOT-count by 1422%. Conversely, for `ctrl_po2`, the CNOT-count improves by 93%, while the T-count worsens by 67%.

Table 4 reports aggregate statistics restricted to the non-negative-gain EPFL instances. Over the full set of 395 EPFL functions, for T-count, the A+D strategy improves 138 instances, leaves 208 unchanged, and worsens 49, while the D+A strategy improves 136 instances, leaves 209 unchanged, and worsens 50. For CNOT-count, A+D improves 90 instances, leaves 214 unchanged, and worsens 91, while D+A improves 89 instances, leaves 217 unchanged, and worsens 89.

As for the Espresso and LGSynth'89 benchmark sets, the aggregate statistics are computed on the non-negative-gain instances using the same metric-specific criterion. Thus, the filtering is applied independently for T-count and CNOT-count, and the decomposed implementation is retained only when it improves, or at least does not worsen, the selected metric after compilation. Percentage-based statistics and non-negative-gain counts are computed only for instances with non-zero baseline cost. Since unchanged instances are included in this subset, the worst non-negative gain is always 0%; therefore, the table reports only the best case and omits a separate worst-case column.

On the non-negative-gain EPFL instances, the aggregate T-count reductions are 21% for both strategies. For CNOT-count, the corresponding aggregate reductions are 59% for A+D and 55% for D+A. These results confirm that the proposed decompositions can be effective as cost-driven preprocessing transformations, provided that the decomposed implementation is retained only when it improves, or at least does not worsen, the selected cost metric after compilation.

5 Conclusion

In this paper, we have considered the class of D-reducible and autosymmetric functions and have shown how these regularities can be exploited to implement compact reversible circuits following both decomposition orders. At the reversible-circuit level, the proposed constructions do not require additional input lines and use three

Just Accepted

Table 4. Aggregate statistics over the non-negative-gain instances from the EPFL benchmark set.

Metric	Strategy	Non-negative cases	Mean gain	Median gain	Aggregate gain	Best case
T-count	A+D	346	9%	0%	21%	max_po127 99%
T-count	D+A	345	8%	0%	21%	max_po127 99%
CNOT	A+D	234	18%	0%	59%	max_po127 100%
CNOT	D+A	234	19%	0%	55%	max_po127 100%

additional output lines: one for the target function and two for the intermediate decomposition results. Moreover, the variable-restoration part is implemented with CNOT gates only and therefore does not contribute any T gates after Clifford+T compilation.

Experimental results, conducted by compiling XAG representations of the decomposed expressions into Clifford+T quantum circuits, have validated the proposed approach. They show that, under the considered cost metrics, the compiled quantum circuits obtained from the D-reducibility–autosymmetry decomposition tend to outperform those obtained from the autosymmetry–D-reducibility decomposition. Future work can consider new regularities to improve quantum compilation. Moreover, we plan to investigate the potential of alternative decomposition techniques to derive more compact quantum circuits.

Another promising direction is the use of the proposed decompositions as local resynthesis transformations for larger logic networks. In this setting, a cut induces a local single-output Boolean function whose inputs are the cut leaves and whose output is the root of the selected cone. The A+D and D+A strategies could then be applied locally to such cut functions, replacing the original cone whenever the decomposed implementation improves the selected quantum cost metric after reversible synthesis and Clifford+T compilation. This perspective is also compatible with AIG-based flows, where each local cut naturally induces a bounded-support Boolean function. Therefore, autosymmetry and D-reducibility could be tested locally on the function represented by the cut before applying the A+D or D+A decomposition. A complete integration of this idea into a cut-based optimization flow, including cut selection and cost-driven replacement, is left as future work.

The synthesis method presented in this paper is formulated and experimentally evaluated for single-output Boolean functions. A first way to use the proposed approach in a multi-output setting is as a local or cut-based resynthesis engine. In this scenario, a large multi-output network can be decomposed into single-output cones or cuts. Autosymmetry and D-reducibility can then be tested locally on these cuts, and the proposed construction can be applied only to those portions of the circuit that exhibit the required regularities. The optimized cuts can then be reinserted into the original network, allowing the method to exploit local regularities while preserving part of the sharing structure of the surrounding multi-output circuit. A more general extension would require defining autosymmetry and D-reducibility directly for multi-output Boolean functions. For autosymmetry, one possible direction is to consider a common autosymmetry space shared by all outputs. For D-reducibility, one should similarly study how the affine spaces associated with different outputs can be combined into a common reduction. However, such global regularities may be weaker than the regularities observed on individual outputs, and additional care is needed to preserve logic sharing during synthesis. We therefore leave the full multi-output formulation and its integration with shared AIG/XAG-based synthesis as future work.

Just Accepted

Acknowledgements

The first two authors are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM), which provided partial support for this work. The last author was supported

by Project "HARMONY – Human-Assisted Robotic Manufacturing for Optimized New Yield" (ID 6161043) funded by PR Lombardia FESR 2021-2027. This publication is part of the grant PID2022-138072OB-I00, funded by MCIN/AEI/10.13039/501100011033/ FEDER, UE, the grant PIPF-2022/COM-24370, funded by the Madrid Regional Government, and part of the grant CEX2024-001471-M funded by MICIU/AEI/10.13039/501100011033.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. *Hypotenuse* 256, 128 (2015), 214335.
[2] Marc Bataille. 2022. Quantum circuits of CNOT gates: optimization and entanglement. *Quantum Information Processing* 21, 7 (2022), 269.
[3] Anna Bernasconi, Alessandro Berti, Valentina Ciriani, Gianna M. Del Corso, and Innocenzo Fulginiti. 2022. XOR-AND-XOR Logic Forms for Autosymmetric Functions and Applications to Quantum Computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 6 (2022), 1861–1872.
[4] Anna Bernasconi, Stelvio Cimato, Valentina Ciriani, and Maria Chiara Molteni. 2020. Multiplicative complexity of autosymmetric functions: Theory and applications to security. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[5] Anna Bernasconi, Stelvio Cimato, Valentina Ciriani, and Maria Chiara Molteni. 2022. Multiplicative complexity of XOR based regular functions. *IEEE Trans. Comput.* 71, 11 (2022), 2927–2939.
[6] Anna Bernasconi and Valentina Ciriani. 2011. Dimension-reducible boolean functions based on affine spaces. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 16, 2 (2011), 1–21.
[7] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. 2006. Exploiting Regularities for Boolean Function Synthesis. *Theory Comput. Syst.* 39, 4 (2006), 485–501.
[8] Anna Bernasconi, Valentina Ciriani, Asma Taheri Monfared, and Stefano Zanoni. 2023. Compact Quantum Circuits for Dimension Reducible Functions. In *2023 26th Euromicro Conference on Digital System Design (DSD)*. IEEE, 776–781.
[9] Anna Bernasconi, Valentina Ciriani, and Licia Monfrini. 2023. Autosymmetric and D-reducible Functions: Theory and Application to Security. In *Advanced Boolean Techniques: Selected Papers from the 15th International Workshop on Boolean Problems*. Springer, 95–110.
[10] Sergey Bravyi and Alexei Kitaev. 2005. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A—Atomic, Molecular, and Optical Physics* 71, 2 (2005), 022316.
[11] RK Brayton. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Springer.
[12] Valentina Ciriani. 2003. Synthesis of SPP three-level logic networks using affine spaces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 10 (2003), 1310–1323.
[13] P M Cohn. 1982. *Algebra: v. 1* (2 ed.). John Wiley & Sons, Chichester, England.
[14] Rolf Drechsler, Alexander Finder, and Robert Wille. 2011. Improving ESOP-based synthesis of reversible logic using evolutionary algorithms. In *European Conference on the Applications of Evolutionary Computation*. Springer, 151–161.
[15] Kenneth Fazel, Mitchell A Thornton, and Jacqueline E Rice. 2007. ESOP-based Toffoli gate cascade generation. In *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE, 206–209.
[16] Craig Gidney. 2018. Halving the cost of quantum addition. *Quantum* 2 (June 2018), 74. doi:10.22331/q-2018-06-18-74
[17] Eiichi Goto and Hidetosi Takahasi. 1962. Some Theorems Useful in Threshold Logic for Enumerating Boolean Functions.. In *IFIP congress*. 747–752.
[18] Pallav Gupta, Abhinav Agrawal, and Niraj K Jha. 2006. An algorithm for synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 11 (2006), 2317–2330.
[19] Jeongwan Haah, Matthew B Hastings, David Poulin, and Dave Wecker. 2017. Magic state distillation with low space overhead and optimal asymptotic input count. *Quantum* 1 (2017), 31.
[20] Ugur Kalay, Douglas V Hall, and Marek A Perkowski. 2000. A minimal universal test set for self-test of EXOR-sum-of-products circuits. *IEEE Trans. Comput.* 49, 3 (2000), 267–276.
[21] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming*. Springer, 486–498.
[22] Robert A Liebler. 2018. *Basic matrix algebra with algorithms and applications*. Chapman and Hall/CRC.
[23] Fabrizio Luccio and Linda Pagli. 1999. On a new Boolean function with applications. *IEEE transactions on computers* 48, 3 (1999), 296–310.
[24] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. 2005. Synthesis of Fredkin-Toffoli reversible networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, 6 (2005), 765–769.
[25] Dmitri Maslov and Ben Zindorf. 2022. Depth optimization of CZ, CNOT, and Clifford circuits. *IEEE Transactions on Quantum Engineering* 3 (2022), 1–8.

Just Accepted

[26] Giulia Meuli, Mathias Soeken, Earl Campbell, Martin Roetteler, and Giovanni De Micheli. 2019. The Role of Multiplicative Complexity in Compiling Low T-count Oracle Circuits. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4-7, 2019*, David Z. Pan (Ed.). ACM, 1–8.
[27] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. 2003. A transformation based algorithm for reversible logic synthesis. In *Proceedings 2003. design automation conference (ieee cat. no. 03ch37451)*. IEEE, 318–323.
[28] Alan Mishchenko and Marek Perkowski. 2001. Fast Heuristic Minimization of Exclusive-Sums-of-Products. *5th International Reed-Muller Workshop* (2001).
[29] Takaaki Mizuki, Taro Otagiri, and Hideaki Sone. 2007. An application of ESOP expressions to secure computations. *Journal of Circuits, Systems, and Computers* 16, 02 (2007), 191–198.
[30] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (2018), 23.
[31] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. 2020. Quantum circuit optimizations for NISQ architectures. *Quantum Science and Technology* 5, 2 (2020), 025010.
[32] Michael A. Nielsen and Isaac L. Chuang. 2016. *Quantum Computation and Quantum Information*. Cambridge University Press.
[33] George Papakonstantinou. 2014. A parallel algorithm for minimizing ESOP expressions. *Journal of Circuits, Systems, and Computers* 23, 01 (2014), 1450015.
[34] KG Papakonstantinou and G Papakonstantinou. 2018. A nonlinear integer programming approach for the minimization of Boolean expressions. *Journal of Circuits, Systems and Computers* 27, 10 (2018), 1850163.
[35] John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.
[36] Heinz Riener, Rüdiger Ehlers, Bruno de O Schmitt, and Giovanni De Micheli. 2020. Exact synthesis of ESOP forms. In *Advanced boolean techniques*. Springer, 177–194.
[37] Mehdi Saeedi and Igor L. Markov. 2013. Synthesis and Optimization of Reversible Circuits - A Survey. *ACM Comput. Surv.* 45, 2 (2013), 21:1–21:34.
[38] Marinos Sampson, Marios Kalathas, Dimitrios Voudouris, and G Papakonstantinou. 2012. Exact ESOP expressions for incompletely specified functions. *Integration* 45, 2 (2012), 197–204.
[39] Tsutomu Sasao. 1993. EXMIN2: a simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued-input two-valued-output functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12, 5 (1993), 621–632.
[40] Tsutomu Sasao. 1993. Logic synthesis with EXOR gates. In *Logic Synthesis and Optimization*. Springer, 259–285.
[41] Tsutomu Sasao. 1996. Representations of logic functions using EXOR operators. *Representations of discrete functions* (1996), 29–54.
[42] Tsutomu Sasao. 2020. On the Minimization of Variables to Represent Partially Defined Classification Functions. In *50th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2020, Miyazaki, Japan, November 9-11, 2020*. IEEE, 117–123.
[43] Tsutomu Sasao. 2024. *Classification Functions for Machine Learning and Data Mining*. Springer Nature Switzerland.
[44] Bruno Schmitt, Mathias Soeken, Giovanni De Micheli, and Alan Mishchenko. 2019. Scaling-up ESOP synthesis for quantum compilation. In *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 13–18.
[45] Eleonora Testa, Mathias Soeken, Heinz Riener, Luca Amaru, and Giovanni De Micheli. 2020. A logic synthesis toolbox for reducing the multiplicative complexity in logic networks. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 568–573.
[46] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. 2020. Efficient and Correct Compilation of Quantum Circuits. In *IEEE International Symposium on Circuits and Systems, ISCAS 2020, Sevilla, Spain, October 10-21, 2020*. IEEE, 1–5.
[47] Saeyang Yang. 1991. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer.

Just Accepted