

Reversible Circuit Synthesis Time Reduction Based on Subtree-Circuit Mapping

Amjad Hawash * , Ahmed Awad and Baker Abdalhaq

Faculty of Engineering and Information Technology, Information and Computer Science Department,
An-Najah National University, PO Box 7, Nablus 00970, Palestine; ahmedawad@najah.edu (A.A.);
baker@najah.edu (B.A.)

* Correspondence: amjad@najah.edu

Received: 30 March 2020; Accepted: 12 June 2020; Published: 16 June 2020



Abstract: Several works have been conducted regarding the reduction of the energy consumption in electrical circuits. Reversible circuit synthesis is considered to be one of the major efforts at reducing the amount of power consumption. The field of reversible circuit synthesis uses a large number of proposed algorithms to minimize the overall cost of circuits synthesis (represented in the line number and quantum cost), with minimal concern paid for synthesis time. However, because of the iterative nature of the synthesis optimization algorithms, synthesis time cannot be neglected as a parameter which needs to be tackled, especially for large-scale circuits which need to be realized by cascades of reversible gates. Reducing the synthesis cost can be achieved by Binary Decision Diagrams (BDDs), which are considered to be a step forward in this field. Nevertheless, the mapping of each BDD node into a cascade of reversible gates during the synthesis process is time-consuming. In this work, we implement the idea of the subtree-based mapping of BDD nodes to reversible gates instead of the classical nodal-based algorithm to effectively reduce the entire reversible circuit synthesis time. Considering Depth-First Search (DFS), we convert an entire BDD subtree in one step into a cascade of reversible gates. A look-up table for all possible combinations of subtrees and their corresponding reversible gates has been constructed, in which a hash key is used to directly access subtrees during the mapping process. This table is constructed as a result of a comprehensive study of all possible BDD subtrees and considered as a reference during the conversion process. The conducted experimental tests show a significant synthesis time reduction (around 95% on average), preserving the correctness of the algorithm in generating a circuit realizing the required Boolean function.

Keywords: reversible circuits; synthesis time reduction; binary decision diagram (BDD); subtrees; depth-first search (DFS)

1. Introduction

With the continuous shrinkage of Very Large Scale Integrated (VLSI) circuits, to keep pace with the performance and small-size requirements caused by the high dependence on personal computers (PCs) in individuals' everyday lives, transistor dimensions are approaching the atomic scale. Therefore, quantum computers are expected to drive forward the advances in VLSI technology in the foreseeable future [1].

Quantum computations are inherently reversible; that is, it is possible to obtain the input vector from an output vector, and vice-versa. With such reversible computations, the number of input bits equals the number of outputs bits, and thus there is no information loss. This loss of information is one of the main sources of energy loss in circuits. The reversible computation avoids bit destruction and thus saves heat loss. Experiments have shown that energy loss can be reduced or even eliminated if computation does not lose information; i.e., if a computation is reversible. Reversible circuits can

be built using reversible gates. Reversible logic computations can be easily applied to the field of quantum computing due to its natural reversibility in computations [2]. In this way, low power design requirements for VLSI chips are achieved [3,4].

Restrictions on the structure of the reversible circuits cause the process of its synthesis to differ significantly from synthesis using traditional irreversible gates. Mainly, fan out and feedback are not allowed; in addition, the input and output bits must be the same, and additional input and output lines might therefore be required to achieve reversibility of computation. Additional inputs are known as constant lines, whereas additional outputs are known as garbage lines [5].

Reversible circuit synthesis has been proposed using numerous algorithms and techniques, each of which improves the synthesis process in terms of speed and optimization [5]. The Boolean function and its companion truth table were the basis for set of these works in the process of synthesizing reversible circuits. However, the Binary Decision Diagram (BDD) forms the basis for other works, in which a reduction in both the time of the synthesis process and the size of the generated reversible circuit is obtained. These works emphasize the BDD size reduction in order to minimize the reversible circuit size, and they were based on the direct mapping between BDD nodes and the corresponding reversible gates in the process of reversible circuit synthesis. Due to the sequential process of this kind of mapping, the one-to-one mapping process consumes some time [6].

In this work, which is an extension of the work presented in [7], we deeply investigate and work on all details related to the idea proposed in the mentioned work as well as implementing the subtree-based mapping of BDD nodes to reversible gates, minimizing the synthesis time of reversible circuits, instead of using nodal-based mapping.

Our contributions can be summarized as follows: (1) Pre-prepared and well-studied combinations of all possible subtrees were created to form a look-up table that maps each possible subtree with its corresponding reversible gates. (2) Suitable hash keys were developed for all investigated possible subtrees in order to make the retrieval time of the corresponding reversible gates constant during the traversal time of a given BDD. (3) We traverse a given BDD (with an inherent Depth-First Search) and convert all its composing subtrees to their suitable reversible gates, forming a complete reversible circuit. (4) A subtree identification algorithm is developed to be used during the traversal time of the BDD subtree. This algorithm investigates a given subtree and constructs a suitable hashing key to be used to directly retrieve the subtree's corresponding reversible gates from the look-up table. (5) More details were added in this work to extensively illustrate the idea presented in the previous work. These details were supported by suitable mathematical formulas and flowcharts. (6) In a previous work, we proposed the idea of minimizing synthesis time, but we did not mention where our work should be considered in the process of converting a Boolean function to a complete working reversible circuit. Here, we illustrated where our work is situated in terms of the steps of the conversion process. (7) A complete code is developed in order to test the work and measure the difference between nodal and subtree-based mapping. (8) Experimental tests were conducted to measure the accurate time of mapping in both nodal and subtree-based cases.

After introducing the work in this section, Section 2 provides several previous works related to the field. In Section 3, we detail some preliminary information about reversible circuits, while Section 4 illustrates their synthesis process. Section 5 discusses the time reduction methodology of the synthesis process. Experimental tests were conducted in Section 6, and finally, the conclusion of the work is presented in Section 7.

2. Related Work

The creation of algorithms that are capable of producing smaller circuits has been the dominant goal of the literature in this field in recent years. We can easily find papers related to the optimization of the process of reversible circuit synthesis. Synthesis time reduction appears as a secondary goal.

Some works have used heuristic algorithms to try to find the best possible solution. For example, the work presented in [8] introduces a synthesis algorithm that uses Covering Set Partitions (CSP),

with no garbage bits. Their work is based on mathematical properties of binary numbers, partially ordered sets and covering graph theory to construct input vectors that are guaranteed to converge. The CSP algorithm can be used to synthesize functions of a large number of variables (up to 30 bits) within a reasonable amount of time.

As a continuation of [8], the authors of [9] compared meta-heuristic algorithms on top of CSP algorithms. Restricted truth tables orders were explored by meta-heuristic algorithms. Genetic and Tabu searches were compared in the work, which also showed that both approaches outperform the typical random search. Quantum cost was used to compare algorithms, which was represented by the number of logical gates.

The authors of the work published in [3] tried to minimize the search space size of the solutions representing feasible reversible circuits by selectively generating permutations for a binary function representing some reversible circuit. Moreover, an Adaptive Genetic Algorithm (AGA) for synthesizing reversible circuits was proposed. The proposed algorithm is fed with a reversible specification of a given circuit and then produces a cascade of Toffoli gates. Several benchmark circuits were tested, and the experimental results showed that the proposed evolutionary algorithm enables optimal or near-optimal solutions with lesser gate counts.

Many recent works have used BDD to synthesize reversible circuits. BDD is a compact data structure that replaces truth tables in the process of building a reversible circuit. Converting binary functions to BDDs before building circuits opens the possibility of using new optimization techniques.

The work presented in [10] is related to the description and evaluation of how optimization techniques for decision diagrams can be exploited for the BDD-based synthesis of reversible and quantum logic. The authors of the work considered shared nodes, complement edges and ordering strategies and presented the gate cascades needed to support these methods. The effect of these techniques on the circuit sizes has been evaluated. The authors found that the BDD optimizations also led to improvements in the circuit sizes.

The effect of intermediate representations on different graph-based methods, such as BDD, inverter graphs and majority inverter graphs, in terms of their usability in making efficient circuit realizations is presented in the work of [11]. Three representations for huge binary functions were compared in this work, and the authors found that BDD is better for small functions due to the compact circuits it generates compared with the other two representations, which showed better improvements in cost compared with BDD with a cost reduction reaching 39% on average. However, effective BDD reordering is not considered in this work.

In the work presented in [12], the impact of different BDD reordering algorithms on the quantum cost of the synthesized reversible circuits was evaluated. This includes greedy, dynamic programming and meta-heuristic based reordering algorithms (Genetic and Simulated Annealing). The experimental results showed that meta-heuristic algorithms present a great deal of evidence for reducing the cost of a synthesized circuit, due to their effective exploration/exploitation in finding the near-optimal BDD size, which was found to be small-sized circuits.

The work conducted in [6] is related to the huge BDD sizes generated from complicated Boolean functions (more than 100 variables). A methodology to derive reversible circuits for a function by a given BDD is proposed in this work. The algorithm used to synthesize the reversible circuits exhibits linear behavior in terms of run-time and space features. Moreover, the BDD size affects the size of the resulting reversible circuit as it transfers the theoretical results known from BDDs to reversible circuits. However, the conducted experimental tests exhibited in better results (circuit cost) and significantly better scalability in comparison to previous synthesis approaches. A set of works have also been conducted regarding the consideration of all possible subtrees of a binary tree. The work presented in [13] is related to the analysis of the total number of subtrees of a tree. The authors of the work studied the number of possible subtrees in a given tree and the characterized set of factors that maximize/minimize the number of subtrees. However, the authors of the work presented in [14] developed algorithms to mine frequent subtrees from a database of rooted trees. They evaluated

the use of two popular sequential encodings of trees to systematically generate and evaluate the candidate patterns. Experimental evaluation showed that their algorithms can achieve up to several orders of magnitude of increased speed for real data-sets when compared to state-of-the-art tree mining algorithms. However, the effectiveness of this algorithm has not been confirmed in the field of reversible circuit synthesis.

Although all of the previous works are related to minimizing the synthesis cost, the work of [15] is related to synthesizing exact minimal reversible circuits for various types of gates and costs. They used the algebraic software GAP to reduce reversible logic synthesis problems. Their group-related algorithm was applied to Peres gates and showed a better performance than for Toffoli gates in the process of increasing the speed of the synthesis of reversible circuits. However, Toffoli gates are simpler to use for manufacturing reversible circuits.

This work is considered to be a continuation of a previous work published in [7]. In that work, we proposed an algorithm to minimize the synthesis time of reversible circuits based on the consideration of subtrees instead of nodal-based mapping between BDD and reversible circuits. In this work, the proposed algorithm is implemented and tested to demonstrate the performance of the proposed algorithm in achieving a significant reduction in the synthesis time while preserving the correctness of the synthesized circuit.

3. Preliminaries and Basic Definitions

The basic logical gates are not reversible by their nature (an exception is the case of a NOT gate). For example, considering AND and OR gates, it is impossible to determine the two input bits from the output one (in the case of AND, the output 0 comes from the inputs 00 or 10 or 01, while for OR, the output 1 comes from the input 11 or 10 and 01). However, the opposite is true with reversible gates: it is easy to determine the input bits from the output and vice versa because the mapping between the input and output vectors is always one-to-one. A reversible gate can be represented by a reversible function with n -bit data for both input and output, where n -bit data are a sequence of bits $x_1, x_2, \dots, x_n$ of length n in which the mapping is bijective function on the set $(0, 1)^n$ of n -bit data on itself [16].

The development of reversible gates and circuits started after Toffoli proposed reversible logic gates in 1977, where a unique input/output association was preserved [17]. Several reversible gates have been proposed, such as NOT, Feynman, Toffoli, Fredkin and Swap gates.

Of the previous circuits, the Toffoli gate became the most important of the reversible gates due to its simplicity of structure. Most of the related works depend on the Toffoli gate in their attempts to optimize reversible circuit synthesis. In this work—as before—we used this gate in our proposal and experiments. The Toffoli gate is a gate with a size of $n \times n$, where the target line flips when all control lines are set to 1; otherwise, the target line signal remains the same. Figure 1 shows a block diagram and schematic representation of a 3×3 Toffoli gate [18] where A and B represent control lines and C represents the target line.

Table 1 represents the truth table that reflects the implementation of the Toffoli gate, where values of A , B (control lines) and C (target line) are inputs while P , Q , and R are the outputs.

The following provides a brief description of the basic terminology behind reversible circuit synthesis in addition to a formulation for its cost metrics [19].

A reversible $n \times n$ function $f : B^n \rightarrow B^n$ is used such that $B \in \{0, 1\}$ is a bijective mapping of every n -bit input vector into an n -bit output vector and any input pattern maps to a unique output pattern. In other words, the output of a reversible function is a permutation of the set of inputs.

Let g be a logic gate that realizes a Boolean function f with n inputs and k outputs. If the function f is reversible, such that $n = k$ with bijective mapping between input and output vectors, the logic gate g is said to be reversible. Notice that additional inputs and/or outputs might be required to be added to make a function f reversible.

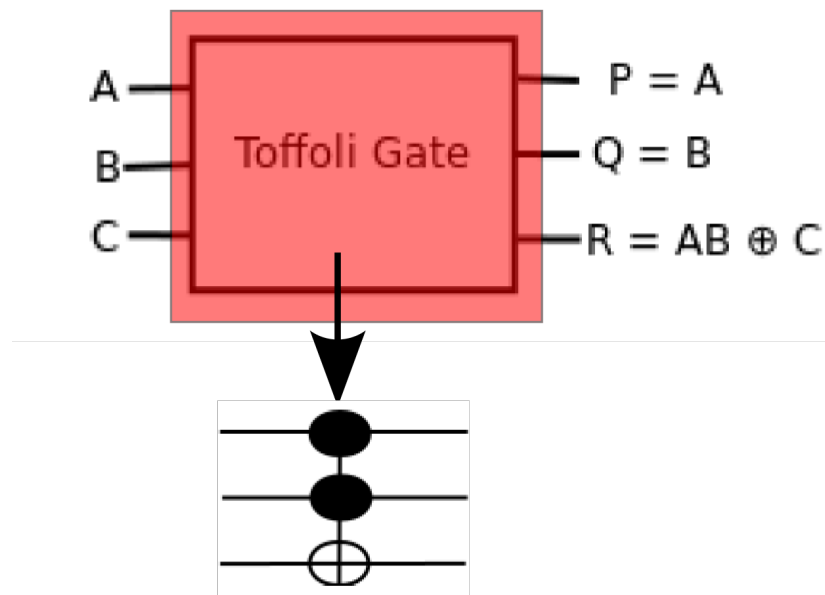


Figure 1. Toffoli gate: block and schematic representations where $\oplus$ represents the target line and $\bullet$ represents the control lines.

Table 1. Toffoli gate truth table.

Input			Output		
A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

A reversible circuit $G = (g_0, g_1, \dots, g_{d-1})$ is a cascade of reversible gates that realize some reversible function. An irreversible function can be embedded in a reversible circuit by adding constant inputs and garbage outputs, as shown in Figure 2, which represents a reversible full adder circuit to find the summation of three input bits (x, y, C_{in}) with the sum output sum and carry out C_{out} . Notice that a constant input of 0 has been added with two garbage (“don’t care”) outputs.

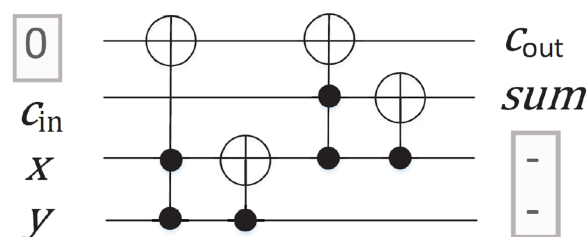


Figure 2. A reversible full added circuit with one constant input and two garbage outputs.

Synthesizing a reversible circuit forms the first step in the design of quantum circuit since the latter is inherently reversible. The reversible gate is mapped into the equivalent functionality in terms of quantum gates. Thereafter, this quantum circuit undergoes an additional procedure to satisfy the required physical constraints and noise elimination. Finally, the quantum circuit undergoes the testing phase [20]. The effort needed to transform a given reversible gate into a set of quantum gates is called the quantum cost, which represents the number of elementary quantum operations needed to realize that reversible gate [21]. This metric is a key concern in the design of quantum circuits. Therefore, it has been used in this paper as an evaluation criterion for the synthesized reversible circuits.

4. BDD-Based Synthesis of Reversible Circuits

Designing reliable and dependable systems that consume minimal amounts of power and energy is considered to be an important challenge to tackle now and in the near future. This can be done by arranging the reversible logical gates in a way that causes the synthesized circuit to transmit signals with high integrity. A set of heuristics for a smart method of reversible logic synthesis has to be met in the process of synthesizing reversible circuits. The following considerations are examples of those that have to be considered by the synthesizing algorithms: (1) the minimization of the output gates or subgates as far as possible; (2) the reuse of these outputs as inputs to other gates; (3) the minimization of garbage signals; (4) the minimization of constants at the inputs of the gates; (5) the creation of an acyclic resulting circuit; and (6) the algorithm must be circuit size-independent and be possible to apply generally [22].

Since fan-out is not permitted, a reversible logic circuit can realize the inverse specification simply by applying the gates in the reverse order. Thus, synthesis can be carried out from the inputs toward the outputs or from the outputs toward the inputs. The literature provides us with a large number of technologies which have been used to synthesize reversible circuits, where both circuit size and synthesis time are of a great consideration. Some of these methods depend on the truth table of a given function as the origin of the synthesis process, while others start from BDD: composition methods, decomposition methods, EXOR logic based methods, genetic algorithms, search methods, group-theoretic methods, the synthesis of regular structures such as nets, lattices, and RPGAs, spectral techniques, exhaustive search and Binary Decision Diagram (BDD)-based methods in which a Boolean function can be represented by BDD [23].

A Binary Decision Diagram (BDD) is a data structure that is used to represent a Boolean function (following Shannon decomposition). It is a directed acyclic graph $G(V, E)$ with a set of nodes where a given node $v \in V$ is either a leaf (with values 0 or 1) or a non-terminal node. The latter node is labeled with one of the input variables of the Boolean function, where Shannon decomposition is applied following Equation (1), where x_i and $\bar{x}_i$ represent a Boolean variable and its complement, respectively. f represents the Boolean function (represented by BDD), f_{x_i} represents the value of the function f when $x_i = 1$ and $f_{\bar{x}_i}$ represents the value of the function f when $x_i = 0$ [24].

$$f = \bar{x}_i f_{\bar{x}_i} + x_i f_{x_i} \quad (1)$$

As an example, Figure 3 illustrates the BDD for the Boolean function $f(x_1, x_2, x_3) = x_1 \oplus x_2 \cdot x_3$, where the symbol $\oplus$ reflects the XOR bitwise operation. However, each path in the BDD starting from the root node until some leaf represents one of the rows in the truth table of the given function represented by the BDD. For example, if we traverse the leftmost path in the graph, we see that the output of the input vector 110 equals 1 where the input vector is $x_1 x_2 x_3$.

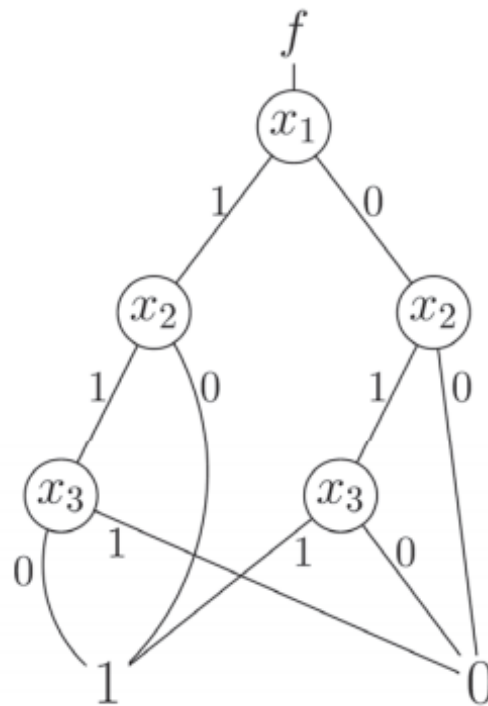


Figure 3. Binary Decision Diagram (BDD) representation of the Boolean function $f = x_1 \oplus x_2 \cdot x_3$ [25].

The size of a BDD is related to the number of the internal nodes it contains (the non-terminal ones). For example, the size of the BDD shown in Figure 3 equals 5. However, in order to reduce the size of the BDD, some reduction rules can be applied without altering its fidelity in representing the required Boolean function. Such rules include shared nodes, complemented edges and BDD node reordering. Notice that BDD reordering should be accompanied by reduction rules to effectively reduce its size. Therefore, both exact and heuristic approaches have been proposed to find the optimal or near-optimal order of the variables in the BDD [26].

The BDD-based algorithm of reversible circuit synthesis implemented in [27] is shown in Figure 4. First of all, the BDD of the input Boolean function is constructed; thereafter, an iterative BDD reordering algorithm followed by the application of BDD reduction rules is applied. Once a small-sized BDD is generated, it undergoes the mapping phase (which is the core of our work), where each node is substituted by a cascade of Toffoli gates following a pre-constructed look-up table. Finally, the synthesized reversible circuit that realizes the input Boolean function is outputted from this algorithm.

In the mapping phase, the BDD is traversed in a Depth-First Search (DFS) fashion, such that each node is substituted into a cascade of reversible gates. The algorithm investigates all variations of BDD nodes and maps each node to its corresponding reversible gates according to a previously prepared look-up table, as shown in Table 2 below. However, this process is a principal contributor to the entire synthesis time due to its linear relation with the number of nodes in the BDD.

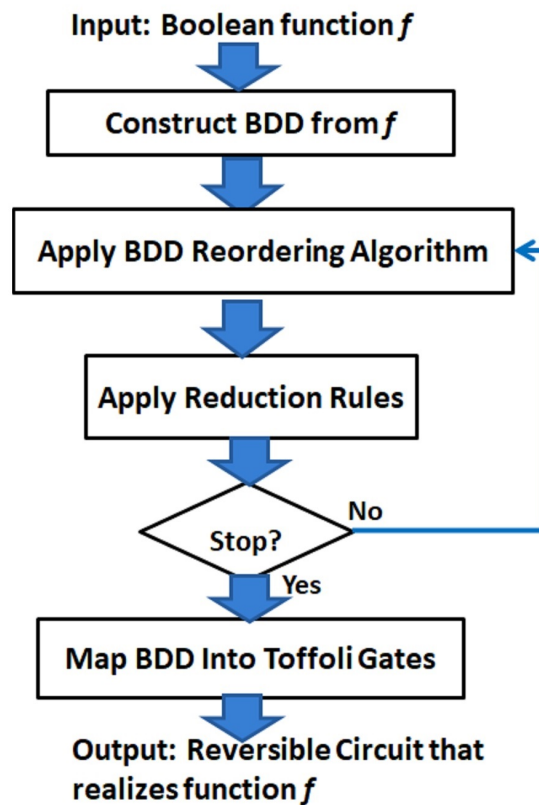


Figure 4. BDD-based synthesis for reversible circuits following the algorithm published in [27].

Table 2. One-to-one mapping from BDD nodes to cascades of reversible gates.

Node	Circuit	Node	Circuit

5. Synthesis Time Reduction

Reducing the time required for synthesizing reversible circuits is essential to reduce the turnaround time for each reversible chip. Most works related to reversible circuit BDD-based synthesis are related to minimizing the quantum cost for a circuit through reversible circuit size minimization; only a minority of works considered the time consumed in the synthesis process [15]. However, the size of the BDD and the size of its resulting circuit depends mainly on the ordering of the corresponding BDD nodes. However, a smaller size of the reversible circuit is not guaranteed by a smaller size of the BDD. Therefore, BDD-based synthesis algorithms for reversible circuits sometimes include the mapping phase in the optimization process for BDD reordering (as shown in Figure 5) which abruptly increases the overall synthesis time [28]. Thus, it is necessary to minimize the time consumed in this mapping phase.

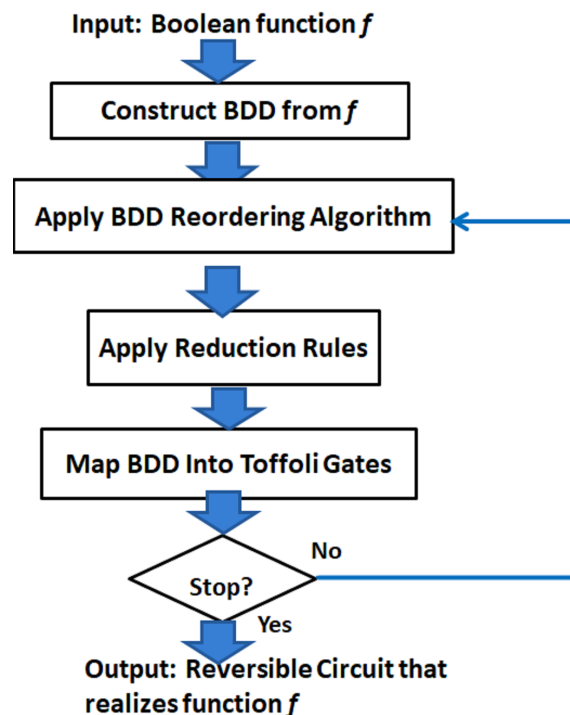


Figure 5. BDD-based synthesis for reversible circuits, integrating the mapping phase in the BDD optimization process.

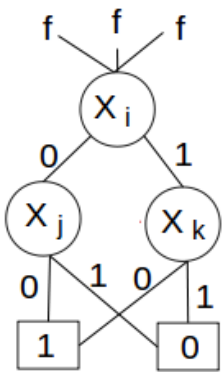
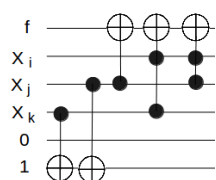
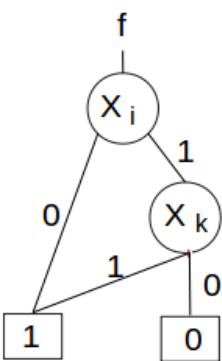
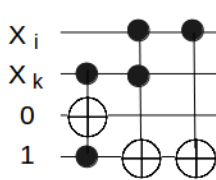
The one-to-one mapping between BDD nodes and their corresponding reversible gates is a time-consuming process, especially in the case of large reversible functions [27]. Because the algorithm has to traverse all nodes in the reduced BDD, the process of constructing the corresponding reversible circuit consumes a large amount of time. Instead of nodal-based mapping, our idea is to consider an entire subtree to be converted directly to a predefined set of reversible gates in the process of reversible circuit synthesis in an attempt to minimize the overall synthesis time. To develop this idea, a new look-up table has to be constructed that maps all possible subtrees to their corresponding reversible gates. This also includes altering the structure of the BDD nodes that form the parents for the considered subtrees by adding a special code to the node that represents the type of the subtree under consideration during the reduction process of the BDD. To do this, and depending on the possible nodes which appear in Table 2, all possible formations of subtrees that could be formed are studied and a new mapping table is constructed. During the traversal of a given BDD, and after studying its nodes, all subtrees are mapped to their corresponding reversible gates as they appear in the new look-up table. The following illustrates the process in detail.

5.1. Look-Up Table Construction

In this work, we implement and test the subtree considered in the process of synthesizing reversible circuits by mapping complete subtrees into their corresponding reversible gates instead of using nodal-based mapping.

Based on the works presented in [6,27], we studied all possible subtrees (35 possible different subtrees were constructed with the possibility of shared nodes) to obtain a complete look-up table of subtrees mapped to their corresponding cascaded reversible gates, taking into account all variations of leaf and non-leaf nodes. This look-up table is constructed one time only as a preprocessing stage and used whenever a circuit needs to be designed based on a BDD for some Boolean function. Each node which appears in the work presented in [6,27] and appears in Table 2 has been given a unique key starting from A to E. After forming all possible permutations of the nodes to form all possible subtrees, the investigation process studies the parent node and the two possible children nodes, taking into account the existence of 0 or 1 on both possible branches of the parent node as well as both possible branches of the two children nodes. Depending on this investigation, we assigned each possible subtree a unique key formed from the keys given to each node and converted each subtree to a set of suitable cascade reversible gates. The result of this process is a list of keys with their corresponding set of reversible gates, all of which are saved in a look-up table. Table 3 below contains two sample elements of the constructed look-up table during this pre-processing stage.

Table 3. Sample subtrees with their mapped reversible gates and assigned hash codes.

Key	Subtree	Circuit
AEE		
BE		
...	...	...

The pseudocode which appears in Listing 1 illustrates the creation of the look-up table discussed above. The code starts by identifying an empty table *lookupTable* and then invokes the method *generatePermutations()* that takes *AllNodes* as a parameter and generates all applicable permutations of nodes (thus forming different subtrees), as shown in Table 2. The reader may notice that not all

nodes are able to be parents for other nodes, such as the last one in Table 2, since this node comes as a leaf only; this obviously limits the number of possible permutations. All possible node permutations (subtrees) are saved in the *Permutations* list as a result of executing the *generatePermutations()* method. The code then continues to evolve all possible subtrees saved in the *Permutations* list to study each subtree and assigns it a suitable unique key by invoking the method *generateKey*, which takes a permutation as a parameter, studies its inbounds/outbounds, parent(s) and children types in order to generate the suitable key for parameterized permutation. In the same loop, both the key and the possible permutation of nodes (subtree) are both saved inside the *lookupTable* by invoking the member method *add* that takes both *key* and *permutation* as parameters. Of course, the construction of the look-up table is done one time only, and used many times later on in the synthesis process.

Listing 1. Look-up table construction pseudocode.

```
List lookupTable = [];
createLookupTable( AllNodes ){
    Permutations [] = generatePermutations( AllNodes );
    foreach( permutation in Permutations ){
        key = generateKey( permutation );
        lookupTable.add( key , permutation );
    }
}
```

5.2. Subtree-Based Mapping Process

In this section, we illustrate the process of the subtree-based mapping process in detail. Given an OUT file (OUT files are textual representations of the reduced BDDs that originally represent some Boolean functions, taken from the Rivkit project as testing materials for our work), we intended to construct the reduced BDD from its OUT file in order to alter the data structure of the nodes to make them suitable for the subtree-based mapping. During the BDD construction, our algorithm marks the nodes that represent the parents of all composing BDD subtrees (the marking process is simply used to update the node data structure with a suitable hash key, as mentioned above), taking into account the fact that each subtree is composed of at most three nodes (one parent and two children). During this process, our algorithm studies these subtrees, taking into account the possibilities that were previously investigated during the look-up table construction illustrated above. As a result, each parent node of the subtrees of the BDD is given a code similar to that given to the similar subtree (the key) during the construction of the look-up table. A link to each parent node is saved in a special table with its corresponding hash key. The result of this process is a table (parent table) containing links to all possible parents (with their corresponding keys) of all composing subtrees for the given BDD.

However, in this work, and because the time consumed in altering the data structure of the parental nodes is beyond the scope of this study since this alteration can be done during the reduction process of BDDs, we intended to execute the alteration process in order to make the BDD structure suitable for our subtree-based mapping process.

We therefore obtain a smaller set of nodes (parent table) as in Figure 6, where each node represents its subtree as well as its corresponding hash key; we can search for the keys of all nodes in the look-up table, and for each matched key, the algorithm adds the corresponding cascade of reversible gates and adds them to the generated reversible circuit. Figure 7 below represents the activities related to the subtree-based mapping process.

For example, Figure 6 represents the BDD for the Boolean function $f(x) = x_1x_2\overline{x_3x_4} + \overline{x_1}x_2x_3x_4 + x_1\overline{x_2}x_3\overline{x_4} + x_1x_2\overline{x_3}\overline{x_4}$, where the dashed, numbered subtrees represent the sequence of their determination in order to convert them into suitable cascade reversible gates. Figure 8 represents the generated reversible circuit of the BDD, where the numbers 1, 2 and 3 associated with a set of reversible gates correspond to the same numbers representing the subtrees in Figure 6.

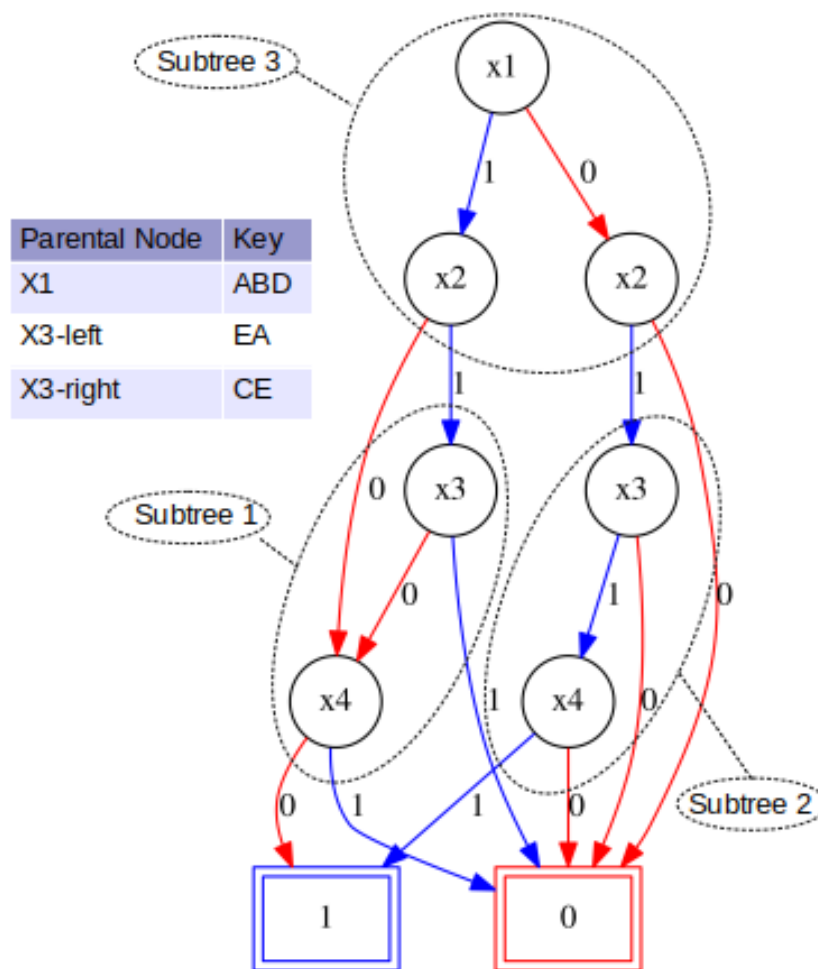


Figure 6. BDD representation of the Boolean function $f(x) = x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2x_3x_4 + x_1\bar{x}_2x_3\bar{x}_4 + x_1x_2\bar{x}_3x_4$, where three subtrees are investigated and the construction of their suitable hash keys was performed as long as their parental nodes were identified.

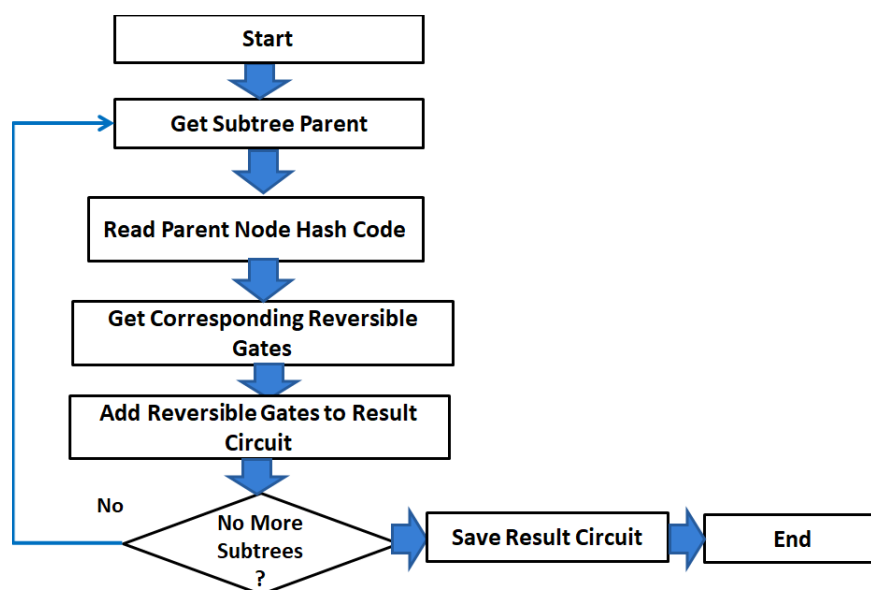


Figure 7. Subtree-based realization of reversible circuits.

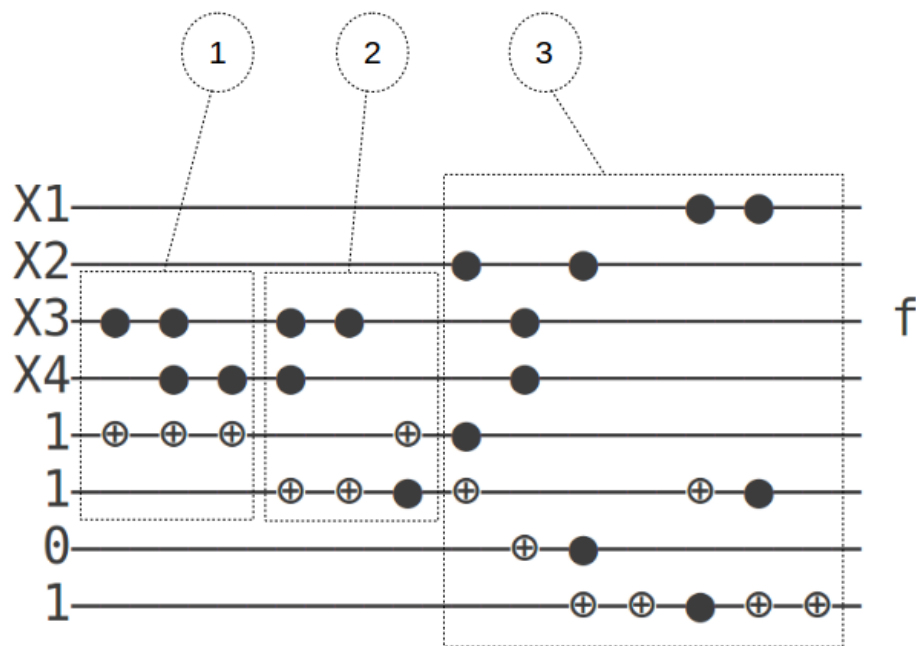


Figure 8. Generated reversible circuit constructed for the BDD in Figure 6 where 1, 1, 0 and 1 are control lines and *f* represents the output of the circuit. The three labels 1, 2, and 3 are related to the investigated subtrees in Figure 6.

The pseudocode which appears in Listing 2 below illustrates the abstracted process of reversible circuit synthesis operation based on subtree mapping. The code starts by calling the method *constructParanetsTable()*, which takes *bdd* as a parameter; using a Depth-First Search method, it visits all BDD nodes, and during the visit, the method decides whether the visited node is a parent for a subtree or only one of the two children that it might have. The output of this method is the table *parentsTable* that contains all subtree parents nodes with their keys. The keys are computed in the same way as that during the construction of the look-up table, taking into account the number of inbound and outbound links. The *Convert()* method is then executed, which takes the parents table *parentsTable* and *lookupTable* as parameters. The loop evolves *parentsTable*, and for each key which appears in the table, the member method *getGates(key)* on the object *lookupTable* is executed; upon a key match, the method adds the cascade of reversible gates to the *gates[]* vector, where the latter is used as a parameter for the member method *insert* on the object *circuit* to add the generated gates to the constructed circuit. Finally, the object *circuit* is returned as a return value from the *Convert* method.

Listing 2. Subtree-based mapping pseudocode.

```
Table parentsTable = constructParanetsTable(BDD bdd);

Convert(parentsTable, lookupTable){
    for each(key in parentsTable){
        gates[] = lookupTable.getGates(key);
        circuit.insert(gates);
    }
    return circuit;
}
```

5.3. Algorithm Run Complexity

The basic operation of the one-to-one based algorithm is to map each BDD node to its corresponding reversible gates (previously Toffoli gates). Conventionally, the algorithm relies on a table that maps each possible case (node type) to its corresponding reversible gate. The size of the table is seven entries, as mentioned earlier. The number of times that one-to-one mapping repeats the mapping process depends on the number of BDD nodes (n). The number of the nodes n depends on the number of variables of the reversible circuit. The worst-case scenario happens when we have an irreducible BDD; in this case, the number of BDD nodes is the same as the number of nodes in a binary tree $n = 2^{v+1} - 1$ nodes, where v represents the number of variables. Thus, we can say that the worst case in the one-to-one algorithm is $2^{v+1} - 1$ operations. The paradigm we used to reduce the algorithm complexity can be categorized as preprocessing. We first prepare a hash table that maps all the possible—up to three nodes—subtrees to their corresponding reversible gates. The preparation of the table is performed only once. The same table can be used to realize any reversible circuit. Therefore, the table can be considered as a fixed part of the algorithm. It does not depend on the input size (in our case, the number of variables). In this case, the size of the look-up table is 35 entries instead of seven entries as in the one-to-one algorithm. This addition of space usage is constant and negligible.

The BDD is divided into subtrees. As described above, each subtree is defined by its parent (or root) node. The parents are defined by starting from the bottom and skipping one level each time. This process can be viewed as creating a new tree of subtrees. The number of levels of the new tree is half of the number of levels of the original tree. The new tree is not binary. Each node on the new tree has four children. The number of resulting nodes—which is the number of subtrees—can be calculated in terms of the number of circuit variables by Equation (2), where s represents the number of subtrees and v again is the number of variables.

$$s = \sum_{k=0}^{\frac{1}{2}v} 4^k = \frac{4^{\frac{v+1}{2}} - 1}{4 - 1} = \frac{2^{v+1} - 1}{3} \quad (2)$$

This results in a reduction of the number of searches using the subtree-based algorithm to three.

The look-up table is a hash table, and so the retrieval speed is constant; i.e., $O(1)$. Therefore, the entry access time using the one-to-one algorithm with seven entries is the same as the access time from the look-up table with 35 entries.

6. Experimental Tests

In order to measure the amount of time reduction by moving from one-to-one to subtree mappings, we conducted an experimental test and recorded the time expended in both approaches. In our test, we used the circuit files that come with the Revkit package (40 files) as testing data. These files contain data regarding the set of BDDs generated by the Revkit. The files contain enough information with respect to the nodes and links of sample BDDs to construct the entire BDD in computer memory. Our algorithm takes these files as an input, works on them to construct the BDD for each file, then converts these BDDs to their corresponding reversible circuits once by considering the one-to-one approach and by considering the subtree. In each approach, the running time (in microseconds) is recorded for the comparison process. The result of the run is a table containing the file name and the running times of both approaches. In order to get rid of the machine dependency and to be sure that results are accurate, we repeated the process for all files 100 times and computed the average time for each file and for each approach.

With respect to benchmark 30, the reduction time is 100% due to the huge size of the related BDD, so the difference between the one-to-one synthesis and subtree-based times is significant. Table 4 contains the collected data (the bold ones represent the most significant results), where the Quantum Cost (QC) has been found to be the same for both one-to-one and subtree-based approaches. This confirms the correctness of the proposed algorithm. Figure 9 depicts the reduction of time between

the two mappings (the time values that appear in Table 4 are normalized here to achieve a better graph visualization).

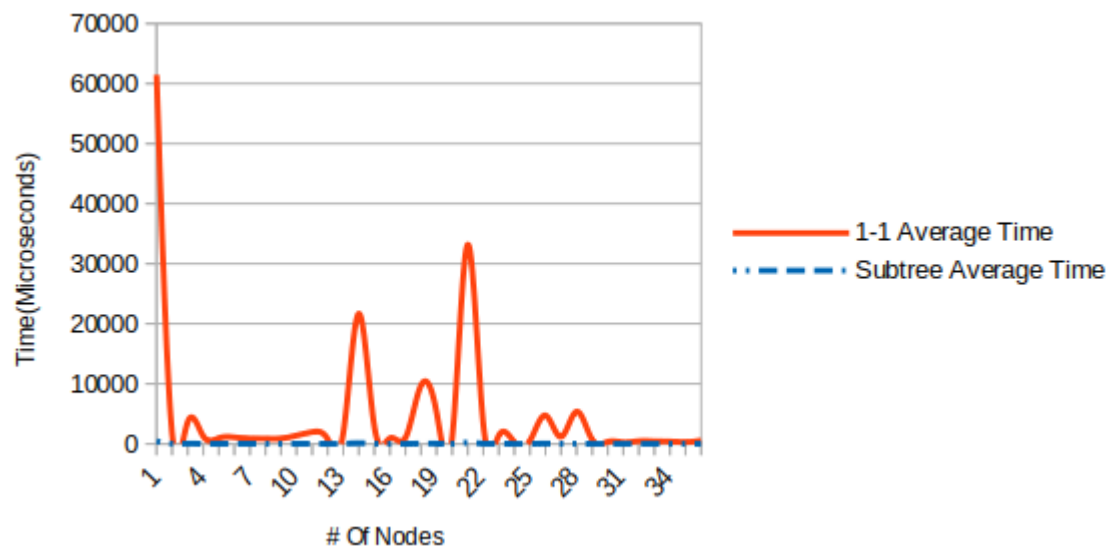


Figure 9. One-to-one vs. subtree time mapping comparison.

Figure 10 illustrates the BDD for one of the benchmarks (Benchmark 9) and its resultant reversible circuit (Figure 11) after applying our proposed algorithm. It is very important to mention that the deeper the BDD, the more Toffoli gates are needed to realize that circuit. It is also important to mention that the source code of our algorithm, along with the benchmarks, is available in [29].

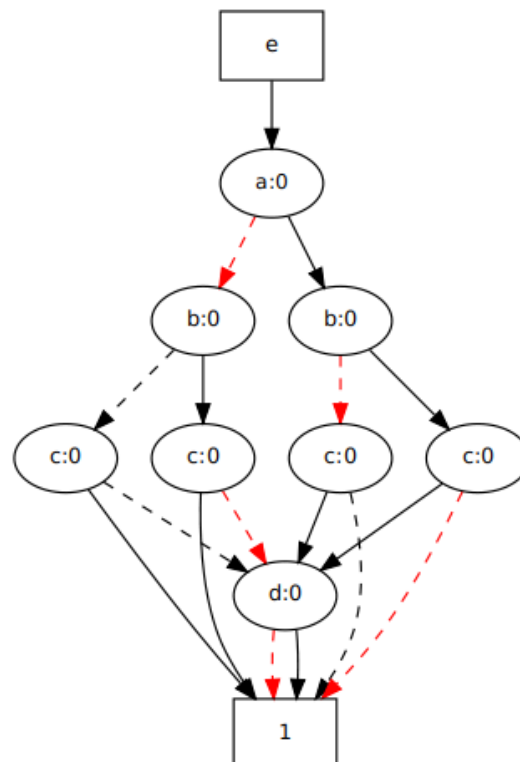


Figure 10. BDD of Benchmark 9.

Table 4. One-to-one vs. subtree mapping time comparison.

Benchmark #	# of Variables	# of Nodes	Quantum Cost (QC)	One-to-One Average Time	Subtree Average Time	Time Reduction Rate
1	62	92	597	61,446.55	440.241	99.28%
2	8	14	41	1669.582	33.902	97.96%
3	16	24	133	3625.391	43.884	98.78%
4	7	11	17	1252.966	18.212	98.54%
5	5	9	5	957.446	15.453	98.38%
6	7	11	17	1087.199	17.875	98.35%
7	6	10	10	899.324	17.656	98.03%
8	7	11	19	869.438	18.085	97.91%
9	6	10	12	889.878	17.659	98.01%
10	8	15	52	1386.826	14.95	98.92%
11	14	21	109	1966.977	15.048	99.23%
12	17	11	206	791.603	10.571	98.66%
13	27	36	206	1760.47	20.612	98.82%
14	138	318	2095	21,679.91	115.8	99.46%
15	39	75	341	2874.424	23.232	99.19%
16	16	24	143	951.772	10.362	98.91%
17	28	32	139	1010.017	10.312	98.97%
18	147	241	2078	9749.405	53.329	99.45%
19	100	139	962	5216.577	38.623	99.25%
20	725	1209	10,197	127,494.95	487.206	99.61%

Table 4. Cont.

Benchmark #	# of Variables	# of Nodes	Quantum Cost (QC)	One-to-One Average Time	Subtree Average Time	Time Reduction Rate
21	11	19	73	978.883	9.333	99.04%
22	5586	7130	64,490	6,614,297.371	1639.358	99.97%
23	562	937	8460	33,181.406	195.1	99.41%
24	2384	2717	24,511	424,256.475	945.057	99.77%
25	157	210	1542	3337.232	48.131	98.55%
26	91	118	965	1407.762	31.061	97.79%
27	10	16	54	274.012	5.526	97.98%
28	39	173	202	521.046	12.91	97.52%
29	152	234	1912	4737.337	51.012	98.92%
30	65,811	131,094	1,112,893	$7.33449651848 \times 10^8$	18,500.089	100.00%
31	89	101	617	1161.141	23.715	97.91%
32	203	394	3399	5378.028	31.978	97.95%
33	51	59	293	687.978	6.219	99.40%
34	22	26	117	349.158	3.273	99.09%
35	13	22	82	260.971	2.371	99.06%
36	37	50	237	456.632	5.08	99.09%
37	49	53	188	429.662	4.43	98.88%
38	27	43	159	390.987	3.625	98.96%
39	16	24	109	288.167	3.016	99.07%
40	53	69	373	538.922	5.366	98.95%

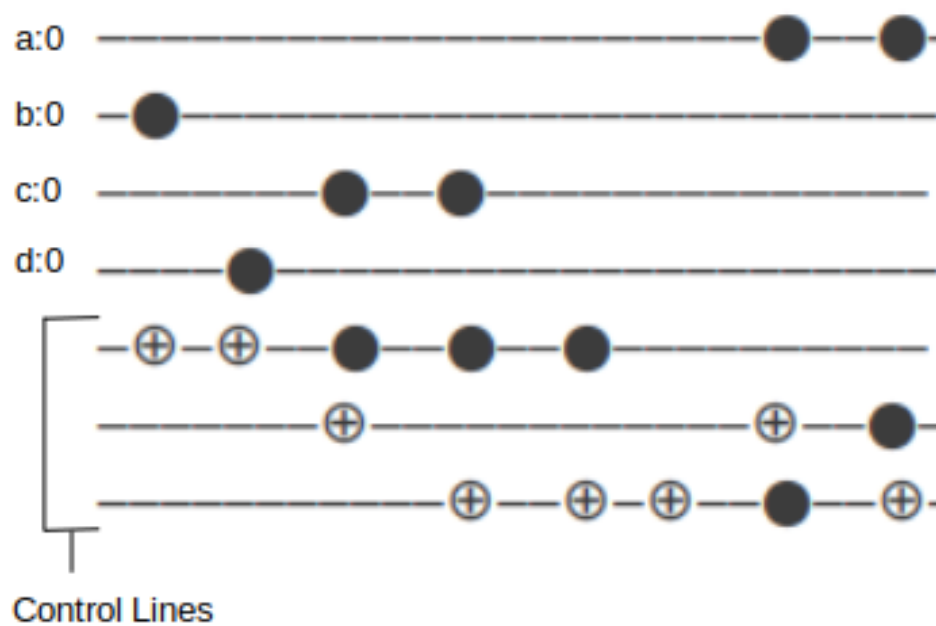


Figure 11. Circuit of Benchmark 9.

7. Conclusions

The time required for reversible circuit synthesis is crucial, especially for huge circuits or those need to be synthesized several times for optimization purposes. In this work, we studied the method of BDD subtree mapping to reversible gates instead of using the nodal-based method. A look-up table has been constructed to map different permutations of possible BDD subtrees to their corresponding cascade of reversible gates. Our algorithm and methodology were examined by experimental tests that gave interesting results. In future works, we are planning to investigate whether the existence of certain subtrees enhances the quantum cost of reversible circuits. In addition, we will consider the reduction of the time required for the performed procedures to satisfy the required physical constraints on the resultant quantum circuit, especially in terms of noise.

Author Contributions: Conceptualization, A.H., A.A. and B.A.; methodology, A.H., A.A. and B.A.; software, A.H.; validation, A.H.; formal analysis, A.H., A.A. and B.A.; investigation, A.H. and A.A.; resources, A.H.; data curation, A.H.; writing—original draft preparation, A.H.; writing—review and editing, A.H., A.A. and B.A.; visualization, A.H. and A.A.; supervision, A.H. and A.A.; project administration, A.H.; funding acquisition, A.H., A.A. and B.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by An-Najah National University under grant number ANN-1920-Sc006.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analysis, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Rubinstein, B.I.P. Evolving quantum circuits using genetic programming. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), Seoul, Korea, 27–30 May 2001; Volume 1, pp. 144–151, doi:10.1109/CEC.2001.934383.
2. Piyush Gau, T. Development of Tool for Mapping Conventional Circuit to Reversible Logic. *arXiv* **2014**, arXiv:1407.0632.
3. Sasamal, T.N.; Singh, A.K.; Mohan, A. Reversible Logic Circuit Synthesis and Optimization Using Adaptive Genetic Algorithm. *Procedia Comput. Sci.* **2015**, *70*, 407–413, doi:10.1016/j.procs.2015.10.054.
4. Landauer, R. Irreversibility and Heat Generation in the Computing Process. *IBM J. Res. Dev.* **1961**, *5*, 183–191, doi:10.1147/rd.53.0183.

5. Mohammadi, M.; Eshghi, M. Heuristic methods to use don't cares in automated design of reversible and quantum logic circuits. *Quantum Inf. Process.* **2008**, *7*, 175–192, doi:10.1007/s11128-008-0081-x.
6. Wille, R.; Drechsler, R. BDD-based Synthesis of Reversible Logic for Large Functions. In Proceedings of the 46th Annual Design Automation Conference, DAC '09, San Francisco, CA, USA, 26–31 July 2009; ACM: New York, NY, USA, 2009; pp. 270–275, doi:10.1145/1629911.1629984.
7. Hawash, A.; Awad, A.; Abdalhaq, B. Towards Reducing Reversible Circuit Synthesis Time. In Proceedings of the 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 9–11 April 2019; pp. 538–543, doi:10.1109/JEEIT.2019.8717492.
8. Hawash, M.; Perkowski, M.; Bleiler, S.; Caughman, J.; Hawash, A. Reversible Function Synthesis of Large Reversible Functions with No Ancillary Bits Using Covering Set Partitions. In Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 11–13 April 2011; pp. 1008–1013, doi:10.1109/ITNG.2011.174.
9. Hawash, M.; Abdalhaq, B.; Hawash, A.; Perkowski, M. Application of Genetic Algorithm for Synthesis of Large Reversible Circuits using Covered Set Partitions. In Proceedings of the Fourth IEEE International Symposium on Innovation in Information Communication Technology (ISIICT 2011), Amman, Jordan, 29 November–1 December 2011; Volume 1.
10. Wille, R.; Drechsler, R. Effect of BDD Optimization on Synthesis of Reversible and Quantum Logic. *Electr. Notes Theor. Comput. Sci* **2010**, *253*, 57–70.
11. Biswal, L.; Das, R.; Bandyopadhyay, C.; Chattopadhyay, A.; Rahaman, H. A template-based technique for efficient Clifford+T-based quantum circuit implementation. *Microelectron. J.* **2018**, *81*, 58–68, doi:10.1016/j.mejo.2018.08.011.
12. Awad, A.; Abdalhaq, B.; Hawash, A.; Johnson, D. A Comparative Analysis of Binary Decision Diagram Reordering Algorithms for Reversible Circuit Synthesis. In Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, 18–21 November 2018; pp. 104–111, doi:10.1109/SSCI.2018.8628765.
13. Li, S.; Wang, S. Further analysis on the total number of subtrees of trees. *arXiv* **2012**, arXiv:1204.6152.
14. Tatikonda, S.; Parthasarathy, S.; Kurc, T. TRIPS and TIDES: New Algorithms for Tree Mining. In Proceedings of the 15th ACM International Conference on Information and Knowledge Management, CIKM '06, Arlington, VA, USA, 6–11 November 2006; Association for Computing Machinery: New York, NY, USA, 2006; pp. 455–464, doi:10.1145/1183614.1183680.
15. Yang, G.; Song, X.; Hung, W.N.N.; Perkowski, M.A. Fast synthesis of exact minimal reversible circuits using group theory. In Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, Los Angeles, CA, USA, 23–25 January 2005; Volume 2, pp. 1002–1005, doi:10.1109/ASPDAC.2005.1466509.
16. Mia, M.; Mukib, M.; Islam, M. An Extended Review on Reversible Logic Gates and their Implementation. *Int. J. Latest Eng. Res. Appl.* **2015**, *1*, 8–18.
17. Sarvaghad-Moghaddam, M.; Orouji, A.A. A new design and simulation of reversible gates in quantum-dot cellular automata technology. *arXiv* **2018**, arXiv:1803.11017.
18. Monz, T.; Kim, K.; Hänsel, W.; Riebe, M.; Villar, A.S.; Schindler, P.; Chwalla, M.; Hennrich, M.; Blatt, R. Realization of the Quantum Toffoli Gate with Trapped Ions. *Phys. Rev. Lett.* **2009**, *102*, 040501, doi:10.1103/PhysRevLett.102.040501.
19. Kanth, B.R.; Krishna, B.M.; Sridhar, M.; Swaroop, V.G.S. A distinguish between reversible and conventional logic gates. *CiteSeerX* **2012**, doi:10.1.1.416.9111.
20. Wille, R.; Chattopadhyay, A.; Drechsler, R. From reversible logic to quantum circuits: Logic design for an emerging technology. In Proceedings of the 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), Samos, Greece, 17–21 July 2016; pp. 268–274.
21. Podlaski, K. Reversible circuit synthesis Using Binary Decision Diagrams. In Proceedings of the 2016 MIXDES-23rd International Conference Mixed Design of Integrated Circuits and Systems, Łódź, Poland, 23–15 June 2016; pp. 235–238, doi:10.1109/MIXDES.2016.7529738.
22. Taha, S. *Reversible Logic Synthesis Methodologies with Application to Quantum Computing*, 1st ed.; 2016th ed.; Springer: Berlin, Germany, 2016, doi:10.1007/978-3-319-23479-3.
23. Yang, C.; Singhal, V.; Ciesielski, M. *BDD Decomposition for Efficient Logic Synthesis*; IEEE: Piscataway, NJ, USA, 1999; pp. 626–631, doi:10.1109/ICCD.1999.808607.
24. Gerez, S.H. *Algorithms for VLSI Design Automation*, 1st ed.; John Wiley & Sons, Inc.: New York, NY, USA, 1999.

25. Wille, R.; Drechsler, R. *Towards a Design Flow for Reversible Logic*; Springer: Berlin, Germany, 2010; pp. 1–184.
26. Brace, K.S.; Rudell, R.L.; Bryant, R.E. Efficient Implementation of a BDD Package. In Proceedings of the 27th ACM/IEEE Design Automation Conference, DAC '90, San Diego, CA, USA, 24–28 June 1990; ACM: New York, NY, USA, 1990; pp. 40–45, doi:10.1145/123186.123222.
27. Drechsler, R.; Wille, R. Synthesis of Reversible Circuits Using Decision Diagrams. In Proceedings of the 2012 International Symposium on Electronic System Design (ISED), Kolkata, India, 19–22 December 2012; pp. 1–5, doi:10.1109/ISED.2012.37.
28. Patel, K.N.; Markov, I.L.; Hayes, J.P. Optimal Synthesis of Linear Reversible Circuits. *Quantum Inf. Comput.* **2008**, *8*, 282–294.
29. Hawash, A.; Awad, A.; Abdalhaq, B. *Reversible Circuits Subtree Mapping Code*. 2020. Available online: <https://github.com/ahmedawad119/Reversible-Circuits> (accessed on 23 May 2020).



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).