Skipping Embedding in the Design of Reversible Circuits

Alwin Zulehner Robert Wille
Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
alwin.zulehner@jku.at robert.wille@jku.at

***Abstract*—Synthesis of reversible circuits finds application in many promising domains but has to deal with the fact that the underlying circuits require a unique mapping from the inputs to the outputs. Existing solutions addressed this problem by additionally performing a so-called embedding process prior to synthesis or by naively mapping building blocks of conventional logic to their corresponding reversible counterparts. This leads to solutions that either suffer from limited scalability or yield circuits with a huge number of additionally required circuit lines. In this work, we conduct investigations to overcome these problems. To this end, we simply ignore the fact that an arbitrary Boolean function to be synthesized might be non-reversible and deal with the resulting problem of ensuring a unique input/output mapping during the actual synthesis process. Experimental evaluations indicate that, following this approach, could provide the basis for an alternative synthesis scheme that allows for synthesizing arbitrary Boolean functions in reasonable time and without the need of a prior embedding process.**

I. Introduction

Reversible circuits employ a computation paradigm in which only bijective functions are realized, i.e. functions that map each input pattern to a unique output pattern allowing for computations from the inputs to the outputs, but also vice versa. This paradigm is particularly suitable for the realization of certain parts of quantum circuits (see e.g. [13] for an overview on quantum circuits and e.g. [2], [14] for overviews on the utilization of reversible circuits in this domain) and received interest due to its promising characteristics with respect to power consumption (motivated by the theoretical considerations conducted by Landauer and Bennett in [8], [3] which recently have experimentally been validated in [4]). Besides that, reversible circuits recently found application in the design of on-chip interconnects [25], encoders [29], or verification [1].

However, synthesis of reversible circuits significantly differs from established design solutions available for conventional circuitry. Due to the required unique mapping, several issues such as the non-existence of fan-out, the requirement that all reversible circuits have to be composed as cascades of reversible gates, and particularly the necessity to realize an arbitrary (and, hence, potentially non-reversible) function in a reversible (i.e. bijective) fashion have to be addressed. In the past, numerous design solutions have been proposed which approach these problems from two different angles as summarized in Fig. 1.

The first scheme (sketched in the top of Fig. 1 and denoted *Functional Synthesis*), employs a two-stage approach: First, the (arbitrary) function to be realized is *embedded* into a reversible one; afterwards, synthesis solutions dedicated to reversible logic are applied to obtain a circuit. Both stages solve problems of significant complexity. More precisely:

- The *Embedding Process* [9], [24], [20], [28] adds further variables to the function (which leads to an exponential growth of the truth table) in order to distinguish non-unique output patterns. To this end, it has to be known how often the function to be realized maps to the same unique output pattern – requiring a consideration of all 2^n possible mappings in the worst case [9][1]. Furthermore, the available degree of freedom for assigning additional variables cannot be exploited during synthesis, since the embedding process yields a fix reversible function.
- The *Actual Synthesis* realizes the embedded function and, by this, all resulting 2^k input/output mappings in terms of a reversible circuit. To this end, various approaches ranging from exact solutions [6] to heuristic solutions e.g. based on truth-tables [16], [11], positive polarity Reed-Muller expansion [7], or Reed-Muller spectra [10] have been proposed. The exact ones are only capable of realizing functions with at most six primary inputs, while the heuristic solutions are capable of realizing functions with up to approx. 30 primary inputs. In order to improve this limited scalability, alternative synthesis approaches have recently been proposed that explicitly exploit efficient data-structures such as decision diagrams [19], [18]. But since all these solutions require an embedded function, they have to cope with the possibly huge overhead introduced by the embedding process.

[1]In fact, it has been shown that this embedding process is coNP-hard [20].

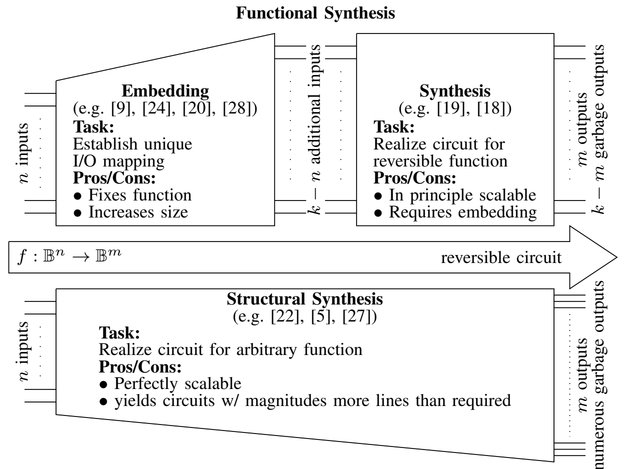


Fig. 1: Reversible circuit synthesis

The second scheme (sketched in the bottom of Fig. 1 and denoted *Structural Synthesis*), aims for directly synthesizing arbitrary Boolean functions. To this end, the function to be synthesized is represented using conventional descriptions such as *Binary Decision Diagrams* (BDDs, [22]), *Exclusive Sum of Products* (ESoP, [5]), or even gate netlists [27]. Afterwards, each building block such as a BDD node, a product/exclusive sum, or a primitive gate is mapped to a functionally equivalent cascade of reversible gates. Since all these building blocks are non-reversible, the equivalent cascade of reversible gates usually requires an additional circuit line (similar to the additional variables in the embedding process). Since, for larger functions, numerous such building blocks are mapped, this yields a number of additionally required lines which is magnitudes larger than the actual minimum (as e.g. evaluated in [24]).

In this work, we conduct investigations to overcome the drawbacks of both schemes. To this end, we simply apply a reversible circuit synthesis algorithm to the function to be synthesized without prior embedding. All problems which occur due to the existence of non-unique output patterns are then handled during the process whenever necessary. This way, problems by non-unique output patterns in particular and non-reversible functions in general are not covered prior to synthesis (as in functional synthesis) or simply ignored (as in structural synthesis), but handled when they matter most: during the synthesis itself.

Experimental evaluations show the promises of this approach. In fact, ignoring embedding and handling the resulting problems during the actual synthesis allows for synthesizing reversible circuits for arbitrary Boolean functions whose number of additionally required circuit lines substantially less than in circuits obtained by structural synthesis. Compared to functional synthesis, the proposed solution is capable of realizing reversible circuits for arbitrary Boolean functions with much more than 30 variables and with significantly lower costs.

The remainder of this work is structured as follows: Section II briefly reviews the applied notation for reversible and non-reversible functions as well as reversible circuits. The main ideas and concepts of the solution proposed in this work are described and illustrated in Section III. Afterwards, details of the resulting implementation are provided in Section IV. Finally, the obtained experimental results are summarized in Section V and the paper is concluded in Section VI.

II. Background

To keep this paper self-contained, this section briefly recaps the applied notation for reversible and non-reversible functions as well as the definition of reversible circuits.

A. Reversible and Non-Reversible Functions

A Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ is called reversible if the mapping from inputs to outputs forms a bijection. Because of this one-to-one mapping, one can determine the input pattern for a given output pattern and vice versa. As this eventually describes a permutation of the input patterns, we can represent each reversible function as a permutation matrix.

Definition 1. *Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ be a reversible function. A* permutation matrix *M representing f is a matrix of dimension $2^n \times 2^n$ in which each column (row) of the matrix represents one possible input pattern (output pattern) of f. The elements $m_{i,j}$, $0 \leq i, j < 2^n$ of the matrix M are defined by*

$$m_{i,j} = \begin{cases} 1 & \text{if } f(j) = i, \\ 0 & \text{otherwise.} \end{cases}$$

Example 1. *Consider the reversible function provided in terms of a truth table as shown in Table Ib. The functionally equivalent permutation matrix is provided in Fig. 3a.*

In this work, we aim for realizing arbitrary (i.e. also non-reversible) functions. To this end, a function matrix representation is applied.

Definition 2. *Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be a non-reversible function. A* function matrix *M representing f is a matrix of dimension $2^k \times 2^k$ with $k = max(n, m)$ in which each column (row) of the matrix represents one possible input pattern (output pattern) of f. The elements $m_{i,j}$, $0 \leq i, j < 2^k$ of the matrix M are defined by*

$$m_{i,j} = \begin{cases} 1 & \text{if } f(j) = i, \\ 0 & \text{otherwise.} \end{cases}$$

Example 2. *Consider the truth table of a half adder (i.e. a non-reversible function) shown in Table Ia. The functionally equivalent function matrix is provided in Fig. 5a.*

Note that each row and each column of a permutation matrix (representing a reversible function), contains exactly one 1-entry (caused by the one-to-one mapping). In contrast, non-reversible functions might map several input patterns to the same output pattern. Hence, the corresponding function matrix might contain rows with multiple 1-entries.

B. Reversible Circuits

Reversible circuits differ from non-reversible ones, since direct feedback and fan-out are not allowed. Typically, they are formed by a cascade of reversible gates. In this paper, we focus on Toffoli gates, a universal type of reversible gates.

Definition 3. *Let $X = \{x_1, \ldots, x_n\}$ be a set of circuit lines. Then, a* reversible circuit *is a cascade of reversible gates $g_1, g_2, \ldots, g_k$. A* Toffoli gate *$g_i = TOF(C_i, t_i)$ is a tuple composed of a set $C_i \subset \{x_i \mid x_i \in X\} \cup \{\overline{x}_i \mid x_i \in X\}$ of positive and negative control lines as well as a single target line $t_i \in X$ with $\{t_i, \overline{t}_i\} \cap C_i = \emptyset$. The value of target line t_i is inverted iff all positive (negative) control lines are assigned one (zero). No other values are affected by the gate.*

In the following, Toffoli gates are considered which are supposed to explicitly invert the value of circuit lines for a given input assignment only. To this end, we introduce the term of a selector set.

Definition 4. *Let $\alpha = \alpha_1 \ldots \alpha_n$ be an input assignment to the lines $x_1, \ldots, x_n$ of a reversible circuit. Then, the set of control lines $C(\alpha) = \{x_i \mid \alpha_i = 1\} \cup \{\overline{x}_i \mid \alpha_i = 0\}$ is the* selector set *of α.*

The costs of reversible gates are measured in terms of quantum costs [2], [12] that depend on the number of control lines, e.g. a Toffoli gate with zero or one control line has quantum cost of 1 and a Toffoli gate with two control lines has quantum cost of 5. In this work, we are applying the metric as introduced in [12].

Example 3. *Fig. 2a shows a reversible circuit composed of 5 Toffoli gates which maps the input pattern 010 to the output pattern 101. This circuit has quantum cost of 9. For a given input assignment $\alpha = 011$, the selector set is $C = \overline{x}_1, x_2, x_3$ i.e. a gate $g = (\{\overline{x}_1, x_2, x_3\}, x_t)$ would invert the target line x_t iff the input α is applied.*

III. Proposed Approach

In this section, we investigate an approach which aims for overcoming the drawbacks discussed in the Section I. The general idea is simple: Instead of conducting embedding prior to synthesis, we propose to simply apply the reversible logic synthesis approaches (i.e. functional solutions summarized in the top of Fig. 1 which are suited for reversible functions only) directly to the non-reversible function to be synthesized and to deal with the problems caused by the non-reversibility during synthesis.

In order to properly describe this idea, we first briefly review the main concepts of two representative synthesis approaches. Afterwards, we discuss the problems which occur if these methods are applied to a non-reversible function (rather than a reversible one) and introduce the proposed solution to overcome those.

TABLE I: Embedding of the half adder function

(a) Non-Reversible

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Reversible

κ	a	b	γ	c	s
0	**0**	**0**	0	**0**	**0**
0	**0**	**1**	0	**0**	**1**
0	**1**	**0**	1	**0**	**1**
0	**1**	**1**	0	**1**	**0**
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	1	1	0	0

TABLE II: Transformation-based synthesis

line	input	output	1st	2nd	3rd	4th	5th
(i)	xyz	xyz	xyz	xyz	xyz	xyz	xyz
0	000	000	000	000	000	000	000
1	001	001	001	001	001	001	001
2	010	101	10**0**	1**1**0	**0**10	010	010
3	011	010	010	010	**1**10	11**1**	**0**11
4	100	110	11**1**	1**0**1	101	10**0**	100
5	101	111	11**0**	1**0**0	100	10**1**	101
6	110	011	011	011	**1**11	11**0**	110
7	111	100	10**1**	1**1**1	**0**11	011	**1**11

A. Representative Synthesis Approaches

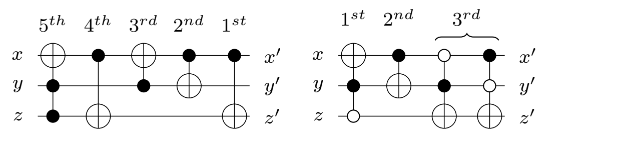
We consider transformation-based synthesis as originally introduced in [11] and QMDD-based synthesis as introduced in [19] as representatives for reversible circuit synthesis approaches. The main idea of these approaches is to determine a sequence of reversible gates $g_1 g_2 \ldots g_k$ that transform the function to be synthesized F to the identity function I. Since $F^{-1} \circ F = I$, reversing the order of the gate sequence to $g_k \ldots g_2 g_1$ realizes F. The function of a half adder (shown in Table Ia) serves as running example. As this obviously is a non-reversible function (the input patterns 01 and 10 map to the same output pattern 01), this function has to be embedded into a reversible one first (e.g. into the one shown in Table Ib; the desired function is highlighted in bold)[2]. This requires the addition of an ancillary variable which yields a constant input κ (constant zero) and a garbage output γ. If κ assumes constant zero, the desired half-adder function is employed while, in general, reversibility is ensured. After this embedding, both considered synthesis approaches become applicable.

Transformation-based synthesis relies on a truth table description of the function to be synthesized. The lines of the truth table are traversed and gates are applied until the output matches the input (i.e. until the identity of both is achieved). Gates are chosen such that already considered lines are not altered. Furthermore, gates are added starting at the output side of the circuit (this is, because *output* values are transformed until the identity is achieved). An example illustrates the idea:

Example 4. *Table II illustrates the respective steps conducted by the transformation-based synthesis. The first column denotes the truth table lines, while the second and third column provides the function to be synthesized (from Table Ib). For brevity, the inputs κ, a, and b as well as the outputs γ, c, and s are denoted by x, y, and z respectively. The remaining columns provide the transformed output values for the respective steps.*

The algorithm starts at truth table line 0*. Since for this line the input is equal to the output (both are assigned to* 000*), no gate has to be applied. The same applies to truth table line* 1*. In contrast, to match the output with the input in truth table line* 2*, the values for z, y and x must be inverted. To this end, three gates $TOF(\{x\}, z)$, $TOF(\{x\}, y)$, and $TOF(\{y\}, x)$ are added as depicted in Fig. 2a – yielding the transformed output patterns as shown in columns 1^{st}, 2^{nd}, and 3^{rd} of Table II, respectively (changed bits are highlighted in bold). Because each gate has a positive control line x or y, this does not affect the previously considered truth table line.*

[2]Recall that already this embedding process is coNP-hard, task [20].



(a) Transformation-based (b) QMDD-based

Fig. 2: Circuits obtained by reversible logic synthesis

In line 3*, gates $TOF(\{x\}, z)$ as well as $TOF(\{y, z\}, x)$ are needed to match the values of x and z (4^{th} and 5^{th} step). For the latter, two control lines are needed to keep the already traversed truth table lines unaltered. Afterwards, all input/output-mappings represent the identity and, hence, the synthesis process has been completed. The resulting circuit, composed of five gates, is shown in Fig. 2a.*

QMDD-based synthesis relies on a matrix representation of the function to be synthesized. Also here, the main idea is to apply gates in order to transform the given function into the identity. From a matrix perspective, the function to be synthesized can thereby be split into four submatrices – one for each quadrant of the original matrix. Then, the identity matrix requires the second and third quadrant to be composed of 0-entries only. This can be achieved by applying Toffoli gates which accordingly swap the respective matrix columns. Recursively applying this scheme to all other submatrices eventually yields the identity matrix and, hence, the desired circuit. Again, an example illustrates the idea.

Example 5. *Consider again the embedded function of a half adder as represented in Table Ib. Fig. 3a provides the corresponding permutation matrix representing the mapping from the inputs (columns) to the outputs (rows)*[3]*. First, this matrix shall be transformed so that the second and third quadrant is composed of 0-entries only. To this end, columns* 010 *and* 110 *must be swapped. This can be accomplished by adding a gate $TOF(\{y, \overline{z}\}, x)$ (the value of x is flipped when $y = 1$ and $z = 0$) yielding the matrix as shown in Fig. 3b. Afterwards, the same scheme is recursively applied to the first and fourth quadrant leading to a swap of columns* 100 *and* 110 *as well as* 101 *and* 111*. Both swaps are accomplished by a single gate: $TOF(\{x\}, y)$; cf. Fig. 3c. Another recursive application of the scheme leads to a swap of columns* 010 *and* 011 *as well as* 100 *and* 101 *(accomplished by $TOF(\{\overline{x}, y\}, z)$ as well as $TOF(\{x, \overline{y}\}, z)$; cf. Fig. 3d). Since the identity matrix has been derived by these swaps, the circuit shown in Fig. 2b realizes the given function.*

Note that working on a permutation matrix (which is of exponential size with respect to the number of inputs/outputs) obviously is not efficient. Hence, the concept illustrated above has been implemented on top of so-called *Quantum Multiple-valued Decision Diagrams* (QMDDs, [15]), which provide an efficient representation for matrices.

B. Applying Non-Reversible Functions

Both synthesis approaches reviewed above can only be applied to reversible functions and thus require a previous embedding process. In this work, we aim for investigating what happens if we ignore the embedding step and directly apply the reversible logic synthesis approach to the non-reversible function. Obviously, this was never intended and leads to serious problems during the synthesis process. We deal with these problems by modifying the function and restore the changes afterwards.

[3]Again, the inputs κ, a, and b as well as the outputs γ, c, and s are denoted by x, y, and z, respectively.

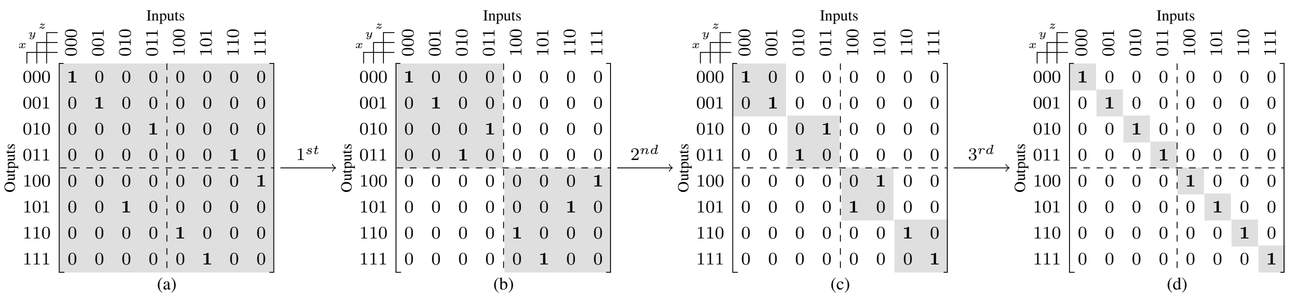


Fig. 3: QMDD-based synthesis

More precisely, whenever a non-unique input/output mapping of f causes problems, we simply flip some of the m output bits in order to resolve the problem. As this alters the function to be synthesized (f) to f', all modifications have to be "buffered" and, after the original synthesis approach terminates, restored. To this end, for each output at position i ($1 \leq i \leq m$), we introduce a new buffer line b_{x_i} which stores all input assignments for which the corresponding output bit has been flipped.

Definition 5. *Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an arbitrary function to be synthesized and let $f' : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be the function which results when conducting a modification as described above. Then,* buffer lines b_{x_i} *are created for $1 \leq i \leq m$ and initially set to 0. Moreover, for all input assignments α for which an input/output mapping has been modified (i.e. for all input assignments α with $f(\alpha) \neq f'(\alpha)$ at bit position i), a gate $TOF(C(\alpha), x_i)$ is added setting the buffer line b_{x_i} to 1*[4].

After the actual synthesis process, these buffer lines can be used to reverse the made modifications. For example, if in a function with three inputs/outputs xyz the mapping from input 010 has been changed from output 101 to output 100, then this modification is stored in a buffer line b_z (which was originally initialized to 0 but, in the considered case with the input assignment $xyz = 010$, set to 1 using the Toffoli gate $TOF(\{\overline{x}, y, \overline{z}\}, b_z)$). After the synthesis has been completed, a gate $TOF(\{b_z\}, z\})$ is applied, which flips the output z back to its intended value.

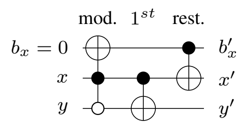
In the remainder of this section, we illustrate the application of this concept to the previously reviewed synthesis approaches. We start with the transformation-based approach:

Example 6. *Consider again the non-reversible function as provided in Table Ia. Applying transformation-based synthesis to this function without embedding leads to the steps as summarized in Fig. 4a. More precisely, nothing has to be done for truth table line 0 and 1 as the inputs are already equal to the outputs. In contrast, the output pattern in line 2 requires to invert both, x and y. But since line 1 and line 2 have the same output pattern (caused by the non-reversibility), any change applied to line 2 will also affect line 1. The original synthesis approach does not work anymore.*

In order to resolve this issue, we modify the function such that the input pattern 10 maps to 11 rather than 01 – leading to the modified output as shown in the fourth column of Table 4a. To store this modification, buffer line b_x (initialized with 0) and the gate $TOF(\{x, \overline{y}\}, b_x)$ are added to the circuit as shown in Fig. 4b. Afterwards, synthesis can continue as usual: The value for y has to be inverted which can be accomplished by a gate $TOF(\{x\}, y)$ (1st step in Fig. 4a and Fig. 4b). This already yields an input/output-mapping representing the identity and, hence, completes the synthesis. Finally, the modification has to be restored, i.e. x has to be inverted for input 10. As this respective case has previously been stored on buffer line b_x, this can easily be accomplished by a gate $TOF(\{b_x\}, x)$, as shown in Fig. 4b.

line (i)	input xy	output xy	mod. xy	1st xy
0	00	00	00	00
1	01	01	01	01
2	10	01	**1**1	1**0**
3	11	10	10	1**1**

(a) Synthesis



(b) Resulting circuit

Fig. 4: Transformation-based synthesis of an arbitrary function

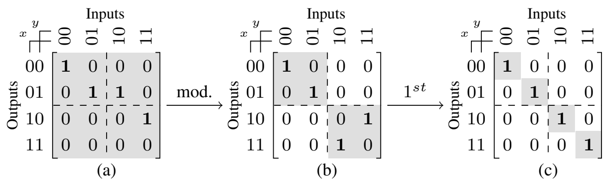


Fig. 5: QMDD-based synthesis of an arbitrary function

The same scheme can be applied to QMDD-based synthesis:

Example 7. *Consider again the non-reversible function represented as function matrix as shown in Fig. 5a. Again, the goal is to perform transformations so that the second and third quadrant is composed of 0-entries only. However, this is not possible by swapping columns anymore. In order to resolve that issue, we modify the function. From a matrix perspective, this means vertically moving 1-entries from one row (here the row for output pattern 01) to another row which does not have a 1-entry yet (here the row for output pattern 11). This is basically the same as discussed before in Example 6 and, hence, requires a buffer line b_x (initialized with 0) and the gate $TOF(\{x, \overline{y}\}, b_x))$ – yielding the matrix as shown in Fig. 5b. Afterwards, synthesis can continue as usual: While the first quadrant already represents an equal input/output-mapping, the forth quadrant requires to swap column 10 and 11. This can be accomplished by adding a gate $TOF(\{x\}, y)$ and completes synthesis. Finally, the modification is restored using the buffer line b_x. This eventually results in the circuit shown in Fig. 4b.*

Overall, we have shown that a realization of arbitrary Boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ can be obtained using reversible circuit synthesis approaches. Although additional circuit lines have to be added dynamically during synthesis for this purpose, their number is bounded by m, since in worst case all of the m output bits have to be flipped. Therefore, at most m buffer lines (one for each output bit) are required.

[4]Recall that $C(\alpha)$ denotes the selector set of α as introduced in Def. 4.

IV. Resulting Implementation

The main concept introduced above can, in principle, be applied to all synthesis approaches for reversible circuits. But since QMDD-based synthesis already proved to be scalable (in contrast to transformation-based synthesis), we provide details on how to employ the proposed concepts for this scheme.

In a first step, the function to be synthesized is represented in terms of a function matrix (cf. Definition 2). Similar to permutation matrices, QMDDs can be applied to efficiently represent the corresponding structures even for large functions. Afterwards, synthesis is performed by swapping columns of the function matrix as illustrated in Section III-B until problems caused by the non-reversibility occur. Then, the function is modified by vertically moving 1-entries in the function matrix to a row which, thus far, is composed of 0-entries only. Of course, these modifications are buffered in respective buffer lines. Afterwards, synthesis and modification steps continue until an identity matrix results. Finally, the modifications are reverted using the buffer lines again.

More formally, synthesis of an arbitrary Boolean function given in terms of a function matrix

$$M = \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix}$$

is conducted as follows:

S1. [Swap columns] Move 1-entries from M_{01} to M_{00} by swapping columns of M until either M_{01} is a zero matrix or each column of M_{00} contains a 1-entry. Since each column of M contains exactly one 1-entry, this automatically moves 1-entries from M_{10} to M_{11} as well. Swapping columns α and β (which differ in bit position i only, i.e. have a Hamming distance of 1) is conducted by applying gate $TOF\,(C(\alpha)\backslash\{\overline{x}_i, x_i\}, x_i)$.

S2. [Vertical movement] If there are any 1-entries in M_{01} (or M_{10}) left, then simply move them to M_{11} (or M_{00}) by flipping the corresponding bit at position i of the output. This modifies the function and has to be buffered. Therefore, moving the 1-entry of column (input) α from row (output) y to y' (by flipping the bit at position i) requires the buffer line b_{x_i} and a gate $TOF\,(C(\alpha), b_{x_i})$.

S3. [Recursive application] After step 2, M_{01} and M_{10} are zero matrices and each column of M_{00} and M_{11} contains exactly one 1-entry, i.e. they are function matrices. Apply Algorithm S recursively to M_{00} and M_{11} to eventually obtain the identity matrix. ∎

Restoring the original function is performed as post-synthesis step which is required to ensure a correct mapping from inputs to outputs, and is conducted by adding one Toffoli gate $TOF\,(\{b_{x_i}\}, x_i)$ for each buffer line b_{x_i}.

V. Experimental Evaluation

In this section, we present the results obtained by the approach described above and compare the measured performance to synthesis schemes considered thus far. To this end, the proposed method has been implemented in C++ on top of *RevKit* [17] – additionally using the BDD package CUDD [21] and the QMDD package made available in [15]. For comparison, we considered BDD-based synthesis [22], QMDD-based synthesis [19], and Ancilla-free synthesis [18]. Furthermore, we compared the number of additionally required circuit lines generated by the proposed approach to the actual minimum (provided in [28]).The numbers for QMDD-based and Ancilla-free synthesis methods have been taken from the corresponding papers. In contrast, results for the BDD-based synthesis have been newly constructed using the publicly available implementation from *RevKit* [17]. The corresponding benchmarks have been taken from *RevLib* [23]. All our experiments have been conducted on a 3.20 GHz Intel i5 processor with 8 GB of main memory running on Linux 4.2.

A. Additionally Required Circuit Lines

As discussed in Section I, the number of additionally required circuit lines is an important criterion in evaluating the performance of synthesis approaches for reversible circuits. While obtaining a circuit requiring a minimal number of additional circuit lines obviously is preferred, determining a corresponding embedding for this purpose is a coNP-hard and, hence, computationally expensive [20]. Accordingly, most proposed solutions that guarantee minimality are rather limited. The currently best known results in this regard have recently been published in [28]. Furthermore, an upper bound of $n + m$ lines is known [24]. In a first series of experiments, we compare the number of additionally needed circuit lines required by the approach proposed in this work to these numbers.

The columns denoted *Lines* in Table III provide the respective numbers, i.e. the minimal value as well as the value obtained by the upper bound (which is $n + m$). Note that knowing the minimal number of required circuit line does not necessarily imply that also an embedding has been determined yet. The number of circuit lines additionally required when applying the synthesis approach proposed in this work is provided in column *Prop.*.

The results clearly show the efficiency of the proposed solution with respect to this metric. Although a minimal number of lines cannot be guaranteed (but are bounded by $n+m$; see Section III), not more than the minimum is required in the majority of the cases (in fact, this holds for 18 out of 31 cases). In many of the other cases just one more line than the minimum is required.

B. Comparison to Existing Synthesis Schemes

In a second series of evaluations, we compared the proposed synthesis approach to existing synthesis schemes. As reviewed in Section I and Fig. 1, two schemes got established in the past: Functional solutions (represented by QMDD-based synthesis [19] and Ancilla-free synthesis [18]), which rely on reversible functions only and structural solutions which rely on conventional functions or circuit descriptions and simply map them to reversible circuits (represented by BDD-based synthesis [22]). While the former relies on embedding and, hence, is not scalable, the latter yields circuits with a substantial number of additionally required circuit lines.

For all these synthesis approaches as well as for the proposed approach, the total number of required circuit lines (denoted by l), the resulting quantum costs (denoted by QC), as well as the run-time needed in order to generate the circuit (denoted by t in CPU seconds) are provided in Table III.

The results clearly show the benefits of the proposed approach: It is scalable and faster by several orders of magnitude (in contrast to the functional solutions) and, at the same time, keeps the number of circuit lines moderate (in contrast to the structural solution). With respect to the quantum costs average improvements of 96.4% and 82.7% compared to the QMDD-based and Ancilla-free approaches are reported, respectively. Indeed, the BDD-based solution performs significantly better with respect to quantum costs, which can be explained by the enormous amount of additional circuit lines that allows for these costs reduction (also observed before in [26]) but eventually yields impractical circuits. Note that the amount of additional circuit lines is more critical, since each line has to be represented physically whereas the quantum cost is an abstract measure to quantify the complexity of the circuit.

Overall, these experimental evaluations confirm that the proposed approach is a promising alternative to previously considered synthesis schemes.

TABLE III: Experimental results

						Functional synthesis					Structural Synthesis					
				Lines			QMDD-based [19]		Ancilla-free [18]		BDD-based [22]			Proposed Approach		
Name	PI	PO	Min.	$n+m$	Prop.	l	QC	t	QC	t	l	QC	t	l	QC	t
C7552	5	16	20	21	21	20	309 008	133.89	180 894	8.53	35	202	0	21	19 310	0.19
bw	5	28	32	33	32	32		TO	3 766 784	2076.51	87	943	0	32	70 512	0.5
sym6	6	1	7	7	7	7	41 552	0.02	8 911	0.11	14	93	0.01	7	2 081	0.08
con1	7	2	8	9	9	8	139 118	0.09	22 988	0.23	16	96	0	9	1 466	0.08
5xp1	7	10	10	17	17	10	724 052	1.07	134 267	3.99	30	254	0.01	17	38 149	0.35
urf2	8	8	8	16	8	8	165 697	0.14	24 066	0.19	209	3 187	0.02	8	88 005	0.13
adr4	8	5	9	13	9	9	290 997	0.26	64 309	0.78	16	74	0.01	9	7 422	0.27
rd84	8	4	11	12	11	11	2 445 223	10.95	401 660	20.6	34	304	0.01	11	34 948	0.08
dc2	8	7	13	15	14	13	5 612 922	74.78	1 395 422	224.06	48	431	0	14	7 157	0.09
misex1	8	7	14	15	15	14	10 115 630	274.82	2 733 073	1046.12	35	288	0.01	15	3 520	0.09
hwb9	9	9	9	18	9	9	629 433	0.91	73 465	0.79	170	2 275	0	9	308 345	0.19
urf1	9	9	9	18	9	9	533 680	0.71	74 858	0.66	374	6 080	0.01	9	243 680	0.18
urf5	9	9	9	18	9	9	185 752	0.13	32 676	0.16	216	2 796	0.01	9	83 765	0.13
sym9	9	1	10	10	10	10	1 276 583	2.94	174 678	3.19	27	206	0.01	10	28 641	0.09
clip	9	5	11	14	14	11	3 036 313	16.88	434 952	18.06	66	704	0	14	95 585	0.13
dk27	9	9	15	18	18	15	35 018 963	3773.21	11 254 565	8598.83	27	140	0	18	5 570	0.12
urf3	10	10	10	20	10	10	1 347 318	3.76	162 225	2.15	668	11 357	0.01	10	544 220	0.33
sym10	10	1	11	11	11	11	3 780 469	23.05	461 538	12.01	32	253	0.02	11	47 761	0.09
urf4	11	11	11	22	11	11	4 508 910	40.74	491 645	12.1	1513	28 523	0.05	11	2 052 585	0.96
Cycle10_2	12	12	12	24	12	12	6 286	0.07	4 200	0.05	39	202	0.03	12	4 966	0.2
co14	14	1	15	15	15	15	38 678 808	2677.82	10 028 634	730.96	27	159	0	15	27 101	0.09
urf6	15	15	15	30	15	15	14 432 936	336.52	1 215 312	3.96	2896	34 361	0.47	15	3 470 000	1.51
exp5p	8	63	68	71	70	∼	∼	∼	∼	∼	206	1 843	0.01	70	1 371 657	4.84
pdc	16	40	55	56	56	∼	∼	∼	∼	∼	619	6 500	0.07	56	5 548 032	11.84
spla	16	46	61	62	62	∼	∼	∼	∼	∼	489	5 925	0.05	62	7 176 407	11.19
cps	24	109	132	133	132	∼	∼	∼	∼	∼	923	8 487	0.02	132	10 762 208	37.71
frg1	28	3	30	31	31	∼	∼	∼	∼	∼	95	747	0.00	31	196 832	0.51
apex2	39	3	42	42	42	∼	∼	∼	∼	∼	498	5 922	0.12	42	4 007 343	8.25
seq	41	35	75	76	76	∼	∼	∼	∼	∼	1617	19 362	0.39	76	66 031 172	432.89
e64	65	65	129	130	129	∼	∼	∼	∼	∼	195	907	0.02	129	40 859 371	82.81
ex4p	128	28	146	156	156	∼	∼	∼	∼	∼	491	3 620	0.01	156	59 708 967	475.45

VI. Conclusions

In this work, we investigated how to overcome the problems of previously proposed solutions for the synthesis of reversible circuits. To this end, we skipped the embedding process (usually a necessity for synthesis of reversible circuits aiming for a small number of additionally required circuit lines) and, instead, dealt with the problem of ensuring a unique input/output mapping during the actual synthesis. Experimental evaluations showed the promises of this alternative direction. In fact, ignoring embedding and handling the resulting problems during the actual synthesis provides the basis for an alternative synthesis scheme that allows for synthesizing arbitrary Boolean functions in reasonable time and without the need of a prior embedding process. Future work includes hardening the findings of this work towards a fully-fledged implementation of this new scheme and further evaluations towards a better understanding of the benefits of this approach.

Acknowledgements

This work has partially been supported by the European Union through the COST Action IC1405.

References

[1] L. G. Amarù, P. Gaillardon, R. Wille, and G. D. Micheli. Exploiting inherent characteristics of reversible circuits for faster combinational equivalence checking. In *Design, Automation and Test in Europe*, pages 175–180, 2016.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[3] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev*, 17(6):525–532, 1973.

[4] A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483:187–189, 2012.

[5] K. Fazel, M. Thornton, and J. Rice. ESOP-based Toffoli gate cascade generation. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 206 –209, 2007.

[6] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple control Toffoli network synthesis with SAT techniques. *IEEE Transactions on CAD*, 28(5):703–715, 2009.

[7] P. Gupta, A. Agrawal, and N. K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Transactions on CAD*, 25(11):2317–2330, 2006.

[8] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961.

[9] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Transactions on CAD*, 23(11):1497–1509, 2004.

[10] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. on Design Automation of Electronic Systems*, 12(4), 2007.

[11] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conf.*, pages 318–323, 2003.

[12] D. M. Miller, R. Wille, and Z. Sasanian. Elementary quantum gate realizations for multiple-control Toffolli gates. In *International Symposium on Multi-Valued Logic*, pages 288–293, 2011.

[13] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[14] P. Niemann, S. Basu, A. Chakrabarti, N. K. Jha, and R. Wille. Synthesis of quantum circuits for dedicated physical machine descriptions. In *Reversible Computation - 7th Int'l Conf., RC 2015, Grenoble, France, July 16-17, 2015, Proceedings*, pages 248–264, 2015.

[15] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler. QMDDs: Efficient quantum function representation and manipulation. *IEEE Transactions on CAD*, 35(1):86–99, 2016.

[16] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *Int'l Conf. on CAD*, pages 353–360, 2002.

[17] M. Soeken, S. Frehse, R. Wille, and R. Drechsler. RevKit: A toolkit for reversible circuit design. In *Workshop on Reversible Computation*, pages 69–72, 2010. RevKit is available at http://www.revkit.org.

[18] M. Soeken, L. Tague, G. W. Dueck, and R. Drechsler. Ancilla-free synthesis of large reversible functions using binary decision diagrams. *J. Symb. Comput.*, 73:1–26, 2016.

[19] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *Asia and South Pacific Design Automation Conference*, pages 85–92, 2012.

[20] M. Soeken, R. Wille, O. Keszocze, D. M. Miller, and R. Drechsler. Embedding of large boolean functions for reversible logic. *CoRR*, 2014.

[21] F. Somenzi. CUDD: CU decision diagram package release 3.0. 0. 2015.

[22] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Automation Conf.*, pages 270–275, 2009.

[23] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: an online resource for reversible functions and reversible circuits. In *International Symposium on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[24] R. Wille, O. Keszöcze, and R. Drechsler. Determining the minimal number of lines for large reversible circuits. In *Design, Automation and Test in Europe*, 2011.

[25] R. Wille, O. Keszocze, S. Hillmich, M. Walter, and A. G. Ortiz. Synthesis of approximate coders for on-chip interconnects using reversible logic. In *Design, Automation and Test in Europe*, pages 1140–1143, 2016.

[26] R. Wille, M. Soeken, D. M. Miller, and R. Drechsler. Trading off circuit lines and gate costs in the synthesis of reversible logic. *Integration*, 47(2):284–294, 2014.

[27] Z. Zilic, K. Radecka, and A. Kazamiphur. Reversible circuit technology mapping from non-reversible specifications. In *Design, Automation and Test in Europe*, pages 558–563, 2007.

[28] A. Zulehner and R. Wille. Make it reversible: Efficient embedding of non-reversible functions. In *Design, Automation and Test in Europe*, 2017.

[29] A. Zulehner and R. Wille. Taking one-to-one mappings for granted: Advanced logic design of encoder circuits. In *Design, Automation and Test in Europe*, 2017.