One-pass Design of Reversible Circuits:
Combining Embedding and Synthesis for Reversible Logic

Alwin Zulehner *Student Member, IEEE,* and Robert Wille *Senior Member, IEEE*

***Abstract*—Reversible computation is a heavily investigated emerging technology due to its promising characteristics in low-power design, its application in quantum computations, and several further application areas. The currently established functional synthesis flow for reversible circuits is composed of two distinct steps. First, an embedding process is conducted which makes non-unique output patterns distinguishable by adding further variables. Then, this function is passed to a synthesis method which eventually yields a reversible circuit. However, the separate consideration of the embedding and synthesis tasks leads to significant drawbacks: In fact, embedding is not necessarily conducted in a fashion which is suited for the following synthesis process. In addition, embedding adds further variables to the function to be synthesized which exponentially increases its corresponding representation in the worst case.**

In this work, we propose one-pass design of reversible circuits, which combines embedding and synthesis. This allows for conducting synthesis with a high degree of freedom, since the embedding that suits best is inherently chosen during synthesis. We propose two solutions (an exact an a heuristic one) following this scheme that improve the currently established synthesis flow by magnitudes in terms of runtime – allowing to synthesize a reversible circuit with a minimum number of lines for some of the frequently considered benchmark functions for the first time. Furthermore, a significant reduction of the costs of the resulting circuits (up to several orders of magnitude) is achieved with this new design flow.

I. Introduction

In contrast to conventional non-reversible logic, the reversible computation paradigm allows for computations not only from the inputs to the outputs, but also from the outputs to the inputs. This paradigm was originally considered in the seminal work by Landauer and Bennett [12], [5], which states that logical reversibility of computations could be one of the keys to power-efficient circuits. Their claim that the information loss caused by non-reversibility is directly related to the power consumption of a circuit has recently been verified experimentally in [6]. Besides that, reversible circuits are also closely related to quantum circuits [19], which promises to solve many tasks significantly faster than conventional logic. Since quantum circuits are inherently reversible, large parts of them can be modeled using classical reversible circuits (as discussed e.g. in [4], [20]). Moreover, the scope of application areas for reversible circuits has grown continuously in the recent years – leading to further utilizations e.g. in the design of encoders [42], on-chip interconnects [32], [35], adiabatic computation [3], [22], or verification [1].

Accordingly, how to efficiently realize reversible circuits has received significant interest. Here, the topology of reversible circuits, which significantly differs from conventional circuitry, poses a particular challenge. To ensure reversibility, these circuits consist of a set of circuits lines which are passed through a cascade of reversible gates. Since this prohibits direct feedback and fan-out, conventional design solutions cannot be utilized and an entirely new design flow is required. In the past, two directions emerged to this end.

Structural Synthesis used (conventional) function and circuit descriptions such as *Binary Decision Diagrams* (BDDs, [31]), *Exclusive-Or Sum-of-Products* (ESoP, [9]), or even gate netlists [38]. Here, 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. As all these building blocks are non-reversible, the equivalent cascade of reversible gates usually requires additional circuit signals (in the domain usually referred to as *circuit lines*). Since numerous such building blocks are mapped for larger functions, this leads to a number of additionally required lines which may be magnitudes larger than the actual minimum (as e.g. evaluated in [34]). Although post-synthesis optimization (e.g. [36], [37]) and adjusted synthesis schemes (e.g. [25]) aimed for reducing the number of circuit lines, the respectively obtained results are still far away from the minimum. Because of this drawback, structural synthesis is not further considered in this work.

As an alternative, *Functional Synthesis* has been proposed. Here, a non-reversible function is embedded into a reversible one prior to synthesis. This *embedding* step [13], [34], [29], [40] adds further variables to the function in order to distinguish non-unique output patterns. Afterwards, the function is passed to the *actual synthesis* method, which eventually yields a reversible circuit. Corresponding synthesis approaches for reversible circuits range from exact solutions [10] to heuristic solutions e.g. based on truth-tables [24], [15], positive polarity Reed-Muller expansion [11], or Reed-Muller spectra [14]. However, since all these approaches rely on an exponential description of the underlying function, they are limited to rather small functions. In order to improve this limited scalability, alternative synthesis approaches have recently been proposed that explicitly exploit efficient data-structures such as decision diagrams [28] or are based on Boolean satisfiability [26].

While these recent improvements lead to impressive progress in the design of reversible circuits with respect to efficiency and scalability, the currently established design flow still suffers from the need to conduct embedding and actual synthesis separately. Because of this, a huge degree of freedom is not exploited since embedding is not necessarily conducted in a fashion which suits the following synthesis step. Furthermore, the embedding step introduces new variables which, in turn, lead to an exponential increase of the size of the function descriptions in the worst case. Later in this paper, Section III

provides a detailed review of these two steps followed by a more detailed discussion of these drawbacks in Section IV.

In this work, we propose an alternative synthesis scheme which addresses these drawbacks. Instead of considering embedding and actual synthesis separately, we introduce the concept of one-pass design of reversible circuits in which both tasks are considered at once – yielding a solution which fully exploits the degree of freedom and avoids the increase in the complexity of the function representation. The proposed one-pass design scheme is applicable to many functional synthesis approaches discussed above. As a representative, we describe and demonstrate the concept by means of the so-called QMDD-based synthesis originally introduced in [28] and recently improved in [39].

An experimental evaluation clearly shows the benefits of combining embedding and synthesis. In fact, for QMDD-based synthesis a significant reduction of the cost of the resulting circuits (up to several orders of magnitude) could be observed. Moreover, even compared to the best currently available synthesis implementation (namely [29] for embedding and [26] for synthesis, both available in RevKit [27]), improvements of 81% on average by means of circuit costs can be achieved. Besides that, a speedup of several magnitudes can be observed – allowing us to synthesize some of the frequently considered benchmarks with a minimum number of circuit lines for the first time.

The remainder of this work is structured as follows: Section II briefly reviews the applied representations for Boolean functions and the basics of reversible circuits. Section III recapitulates the currently established two-stage design flow. Section IV discusses the drawback of the sole consideration of these two steps and introduces the proposed one-pass design of reversible circuits. Finally, the obtained experimental results are summarized in Section V while the paper is concluded in Section VI.

II. Background

We briefly recapitulate the basics of Boolean functions and their representation as well as reversible circuits in this section.

A. Boolean Functions and Their Representation

Boolean functions can be represented by truth tables (as e.g. shown in Fig. 1a). For the synthesis schemes described in this work, we also use *function matrices* to represent Boolean functions.

Definition 1. *Let $f : \mathbb{B}^n \to \mathbb{B}^m$ be a Boolean function. Then, the* function matrix *M of f is a $2^k \times 2^k$ matrix with $k = \max(n, m)$ and elements $m_{i,j}$, $0 \leq i, j < 2^k$ such that*

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

The columns (rows) of a function matrix represent the inputs (outputs). If an input maps to an output, the corresponding entry of the matrix is set to 1. All other entries in the function matrix are set to 0. Consequently, each column of a function matrix contains at most one 1-entry. In contrast, a row may contain multiple 1-entries, because more than one input combination may map to the same output pattern.

x_2	x_1	x_2'	x_1'
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	1

(a) Truth table

Outputs \ Inputs (x_2 x_1)	00	01	10	11
00	**1**	0	0	0
01	0	**1**	0	**1**
10	0	0	**1**	0
11	0	0	0	0

(b) Function matrix

Fig. 1: Representations for a Boolean function

x_2	x_1	x_2'	x_1'
0	0	1	1
0	1	1	0
1	0	0	0
1	1	0	1

(a) Truth table

Outputs \ Inputs (x_2 x_1)	00	01	10	11
00	0	0	**1**	0
01	0	0	0	**1**
10	0	**1**	0	0
11	**1**	0	0	0

(b) Permutation matrix

Fig. 2: Representations for a reversible function

Example 1. *Fig. 1a shows the truth table of a Boolean function f. The corresponding function matrix representation is depicted in Fig. 1b. For example, the fourth column of the function matrix represents the input 11 which, according to f, is supposed to map to the output* 01*. Hence, the fourth column (representing input* 11*) contains its 1-entry in the second row (representing output* 01*).*

Reversible functions are a subset of Boolean functions, defined as follows.

Definition 2. *A Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ is* reversible*, if $n = m$ and the function is a bijection.*

For reversible functions, each column and each row of the function matrix contains exactly one 1-entry, because there is a unique mapping from inputs to outputs and vice versa. Consequently, the function matrix of a reversible function is a *permutation matrix*.

Example 1 (continued)**.** *Fig. 1b shows the function matrix of a non-reversible function. The non-reversibility can be seen in the second row (output 01) of the matrix: two 1-entries in a single row. These entries are in the second and fourth columns and, therefore, represent input combination 01 and 11, respectively. Additionally, there are only 0-entries in the last row of the function matrix, which also is a violation of reversibility, because no input combination is mapped to output 11.*

In contrast, in the truth table shown in Fig. 2a each output pattern occurs exactly once. Since the number of inputs is also equal to the number of outputs, this truth table describes a reversible function. Fig. 2b shows the corresponding permutation matrix, which contains exactly one 1-entry in each row and in each column.

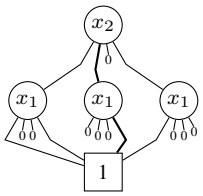


Fig. 3: QMDD representation of the function matrix of Fig. 1b

B. Quantum Multiple-Valued Decision Diagrams (QMDDs)

In this section, we review how to efficiently represent function matrices for large functions. To this end, we recapitulate so-called *Quantum Multiple-Valued Decision Diagrams* (QMDDs) introduced in [16], [21]. QMDDs allow for the efficient representation and manipulation of quantum computations – a future way of doing computations which relies on quantum-mechanical characteristics such as dedicated quantum bits, superposition, entanglement, etc. (see [19] for details). Since quantum operations are represented by square (unitary) matrices of dimension $2^n \times 2^n$, this type of decision diagram is also suited for representing function matrices as considered in this work. For simplicity, we neglect the quantum-related issues since they are not necessary for the tasks considered in this paper.

To efficiently represent a function matrix M, QMDDs provide a decision diagram structure where each node partitions the matrix according to a variable x_i ($n \geq i \geq 1$). More precisely, let's assume x_n is the most significant variable of M. Then, the matrix can be decomposed into four sub-matrices, where each of them represents one of the four possible mappings of x_n, i.e.

- from 0 to 0 (left upper sub-matrix; denoted $M_{0\to0}$),
- from 1 to 0 (right upper sub-matrix; denoted $M_{1\to0}$),
- from 0 to 1 (left lower sub-matrix; denoted $M_{0\to1}$), and
- from 1 to 1 (right lower sub-matrix; denoted $M_{1\to1}$).

Each of these sub-matrices is again represented in terms of a node in the decision diagram. By recursively continuing this partition, smaller sub-matrices result until a single value (i.e. a terminal) is reached. Since the resulting sub-matrices often include a significant amount of redundancy or are even identical, sharing is possible (similar to other decision diagrams such as BDDs [7]). Moreover, zero sub-matrices (i.e. matrices which are solely composed of 0's) frequently occur which, independently of their dimension, can simply be represented by a 0-stub. This eventually allows for a rather compact representation.

Example 2. *Fig. 3 shows the resulting QMDD representation of the matrix shown in Fig. 1b. Note that the successors of a node, i.e. the first, second, third, and fourth edge, represent $M_{0\to0}$, $M_{1\to0}$, $M_{0\to1}$, and $M_{1\to1}$, respectively.*

As an example, the bold path to the 1-terminal[1] *denotes that the mapping of the variables x_2 and x_1 from 1 and 1 to 0 and 1, respectively, is a valid input/output mapping of the function represented by the QMDD. On the other side, no mapping e.g. of the variable x_2 from 0 to 1 exists (as represented by $M_{0\to1}$ terminating in a 0-stub).*

[1]For brevity, paths to the 1-terminal are called 1-paths in the following.

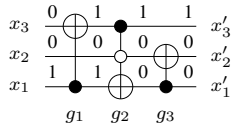


Fig. 4: Reversible circuit

Using QMDDs as introduced above allows for the compact representation of function matrices.

C. Reversible Circuits

Circuits realized in reversible logic are structurally different from conventional ones. They do not directly allow feedback and fan-out and, hence, need to be described by n circuit lines which are passed through a cascade of gates [8], [23]. Moreover, in order to ensure reversibility, each gate must realize a reversible function. In the domain of reversible logic, the Toffoli gate is one of the most established gate types as it is reversible and universal, i.e. every reversible function can be realized with Toffoli gates only.

Definition 3. *Let $X = \{x_n, \ldots, x_2, x_1\}$ be a set of circuit lines. Then, a* reversible circuit *is a cascade $G = g_1 g_2 g_3 \ldots g_h$ of h reversible gates g_i. A* reversible gate *(here:* Toffoli gate*) $g_i = TOF(C_i, t_i)$ consists of a set $C_i \subseteq \{x_j \mid x_j \in X\} \cup \{\overline{x}_j \mid x_j \in X\}$ of positive (x_j) and negative ($\overline{x}_j$)* control lines *and a* target line *$t \in X$ with $\{t_i, \overline{t}_i\} \cap C_i = \emptyset$. A circuit line cannot be used as positive and negative control at the same time. The value of the target line t_i is inverted iff the values of all positive control lines $x_j \in C_i$ evaluate to 1 and all negative control lines $\overline{x}_j \in C_i$ evaluate to 0. All lines other than the target lines pass through the gate unchanged.*

In the following, positive control lines, negative control lines, and the target line of a Toffoli gate are depicted using symbols ●, ○, and ⊕, respectively.

Example 3. *Fig. 4 shows a reversible circuit composed of three circuit lines and three Toffoli gates. Furthermore, the circuit is labeled with the values on the circuit lines for input $x_3x_2x_1 = 001$. The first gate $g_1 = TOF(\{x_1\}, x_3)$ inverts the value of the target line x_3 since the positive control line x_1 is assigned 1. For the same reason (control lines are accordingly assigned), the second gate $g_2 = TOF(\{x_3, \overline{x}_2\}, x_1)$ inverts the value of the target line x_1. In contrast, the third gate $g_3 = TOF(\{x_1\}, x_2)$ does not invert the value of the target line x_2, because the positive control line x_1 evaluates to 0.*

The complexity of reversible circuits are usually measured in terms of quantum costs, i.e. the cost of the reversible circuits when transformed into a quantum circuit. The quantum cost of a reversible gate depends on the number of control lines as well as on the on the library of quantum gates, the reversible circuit is mapped to. A commonly used library is the *NCV* library, where the quantum cost is determined by

the number of resulting quantum gates [4], [17]. Recently, also the *Clifford+T* library has gained importance. Using this library, the quantum cost is determined by the number of *T-gates* that have to be processed sequentially (and, therefore, denoted *T-depth*) [2]. The quantum cost of reversible circuits constantly changes, because better mappings to the respective gate library are determined continuously. Therefore, we use the most recent cost function provided in RevKit [27] which determines the costs of a reversible circuit in terms of T-depth.

III. Functional Reversible Circuit Synthesis

In this section, we review functional synthesis of reversible circuits. As discussed in Section I, this process is composed of two steps. First, the function to be synthesized has to be embedded into a reversible one. Then, a synthesis approach is used to realize the reversible function as a reversible circuit. In Section III-A, we discuss the embedding process in detail. In Section III-B, we review the synthesis process (using a solution based on QMDDs, which was originally proposed in [28] and recently improved in [39], as a representative). Throughout the whole section, we use the Boolean, non-reversible function shown in Fig. 1a as running example.

A. The Embedding Process

We consider Boolean functions $f : \mathbb{B}^n \to \mathbb{B}^m$ with n primary inputs and m primary outputs. If f is non-reversible, multiple input combinations might be mapped to the same output pattern. Furthermore, it might be the case that there exist no input combination that maps to a certain output pattern. Since reversibility requires a unique mapping from inputs to outputs, non-unique output patterns must be made distinguishable. To this end, additional outputs – so called *garbage outputs* – are added to the primary outputs. Assuming that the most frequent output pattern occurs μ times, $k = \lceil \log_2 \mu \rceil$ additional outputs are required to distinguish all occurrences of this pattern [13], [34], [29], [40].

Example 4. *Consider the non-reversible function depicted in Fig. 1a. Since 01 is the most frequent output pattern and occurs twice, at least $k = \lceil \log_2 2 \rceil = 1$ garbage output is required to embed the function.*

The addition of garbage outputs results in extra columns in the truth table. Since we are not interested in the value of the garbage outputs, they can arbitrarily be assigned. However, there are dependencies when assigning the garbage outputs: they have to be chosen in such a way, that they are assigned differently for all occurrences of an output pattern. In the following, this dependency is represented by an asterisk (*).

Example 4 (continued)**.** *The value of the garbage outputs of input patterns 01 and 11 depend on each other, because these inputs map to the same output. As soon as the garbage output for one of the input patterns is fixed to 1 (0), the garbage output for the other input pattern must be fixed to 0 (1) to ensure reversibility.*

In addition to a unique mapping from inputs to outputs, reversibility requires that the number of inputs and outputs has to be equal. Therefore, if n is larger than $m + k$, $n - (m + k)$ further garbage outputs are added and marked with *. In the opposite case, $m + k - n$ additional inputs (so called *ancillary inputs*) have to be added to the function – resulting in a function with $\max(n, m + k)$ inputs and outputs in both cases. Each additional input doubles the number of rows in the truth table. If all ancillary inputs are assigned 0, the reversible function evaluates to the originally specified output. For all other assignments to the ancillary inputs, again arbitrary output values can be applied – even for the primary outputs. Note that also here dependencies have to be considered. In fact, while all outputs are don't care in these cases, each truth table line must still be unique in order to ensure reversibility. In the following, this is represented by a dot (·).

TABLE I: Embedding of a non-reversible function

(a) Degree of freedom

x_2	x_1	a	x_2'	x_1'	g
0	**0**	0	**0**	**0**	*
0	0	1	·	·	·
0	**1**	0	**0**	**1**	*
0	1	1	·	·	·
1	**0**	0	**1**	**0**	*
1	0	1	·	·	·
1	**1**	0	**0**	**1**	*
1	1	1	·	·	·

(b) One possible embedding

x_2	x_1	a	x_2'	x_1'	g
0	**0**	0	**0**	**0**	0
0	0	1	1	1	0
0	**1**	0	**0**	**1**	0
0	1	1	1	1	1
1	**0**	0	**1**	**0**	1
1	0	1	0	0	1
1	**1**	0	**0**	**1**	1
1	1	1	1	0	0

Example 4 (continued)**.** *One ancillary input is needed to ensure that the number of inputs is equal to the number of outputs. This yields an extended truth table as shown in Table Ia. If the additional input a is set to 0, the intended function can be obtained from the outputs x_2' and x_1'. The values of the additional garbage output g as well as for the remaining input assignments (i.e. for $a \neq 0$) can arbitrarily be chosen (represented by * and ·, respectively) as long as the dependencies discussed above are considered.*

Finally, the embedding process is completed by assigning precise values to all entries represented by * and · while considering the discussed dependencies.

Example 4 (continued)**.** *Assigning the *- and ·-entries in the truth table shown in Table Ia so that a reversible function results can be conducted in various fashions. More precisely, the *-entry in each of the first, the third, and the fifth row of the truth table can be assigned in two ways (either 0 or 1). In contrast, there is no choice for assigning a value to the *-entry of the seventh row, since its value has to be the negation of the value we assigned to the *-entry in the third row. Hence, there are $2 \cdot 2 \cdot 2 \cdot 1 = 8$ possibilities for assigning precise values to the $*$-entries. The remaining four output combinations (including primary outputs as well as garbage outputs) can then be assigned to the truth table rows consisting of ·-entries in $4! = 24$ different fashions. Consequently, $8 \cdot 24 = 192$ possibilities exist for assigning * and · with precise values – a quite large degree of freedom. One of the 192 possibilities is shown in Table Ib.*

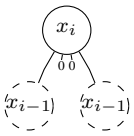


Fig. 5: Identity structure

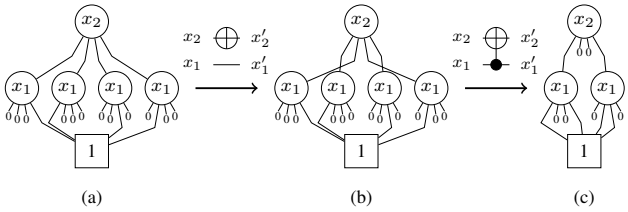


Fig. 6: Effects of applying Toffoli gates to QMDDs

B. QMDD-based Reversible Circuit Synthesis

The embedding process described above yields a reversible function which can be represented in terms of a permutation matrix M. Then, the synthesis task is to determine a reversible circuit $G = g_1 g_2 g_3 \dots g_h$ (as reviewed in Section II-C) which realizes M. This can be conducted by applying reversible gates g_i to M so that, eventually, the identity matrix I results. Let's assume a cascade of reversible gates $G^{-1} = g_h g_{h-1} \dots g_2 g_1$ applied to M transforms M into I. Then, due to the reversibility ($M \circ M^{-1} = I$), the inverse cascade $G = g_1 g_2 g_3 \dots g_h$ realizes M.

This leaves the question how to efficiently determine the gates needed in order to transform M to I. As discussed in Section II-B, permutation matrices can be represented efficiently by QMDDs. Since the identity matrix I only represents mappings from 0 to 0 and from 1 to 1, each node in the QMDD has to be transformed such that its second edge and third edge point to a 0-stub (as illustrated in Fig. 5 for a node labeled x_i). Hence, for a given QMDD M, the task remains how to apply reversible gates so that eventually this structure results.

This task is addressed by successively transforming the QMDD towards the identity. To this end, the nodes of the QMDD are considered in a breadth-first traversal from the top to the bottom. In each step, the currently considered node (representing the partition according to variable x_i) is transformed into the desired structure. This is accomplished by applying Toffoli gates which move all 1-paths of the second and third edge to the first and fourth edge – eventually leading to nodes as illustrated in Fig. 5.

The main principle is to use Toffoli gates to swap QMDD-paths. More precisely, applying e.g. a gate $TOF(C, x_i)$ to a given QMDD inverts the input of the mapping of variable x_i for all paths represented by C. This way Toffoli gates may be used to swap e.g. a mapping from 0 to 1 to a mapping from 1 to 1 and, by this, moving 1-paths from the third edge to the fourth edge – bringing it closer to the identity structure. Again, an example illustrates the idea.

Example 5. *The gate $TOF(\emptyset, x_2)$ inverts the value of the input of variable x_2 for all paths and, therefore, simply exchanges the first (third) and the second (fourth) edge of the root node of the QMDD shown in Fig. 6a. The resulting QMDD is depicted in Fig. 6b. The gate $TOF(\{x_1\}, x_2)$ inverts the input of variable x_2, but only for paths where variable x_1 maps from 1 to anything. This already yields the identity structure for the root node of the QMDD as shown in Fig. 6c.*

In addition, we have to make sure that applying Toffoli gates does not affect previously traversed nodes. This can be accomplished by adding control lines to each Toffoli gate which specify the path to the currently considered node.

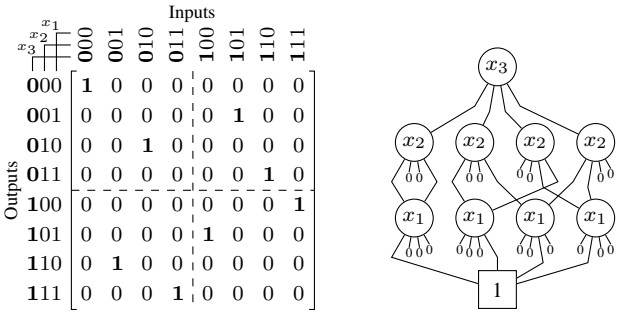
Example 6. *Consider the QMDD shown in Fig. 6c and assume that the rightmost QMDD node with label x_1 is currently processed. Each gate applied for processing this node has to include a positive control line x_2. As a result, only paths with $x_2 = 1$ are swapped, i.e. paths which run through the fourth edge of the top node (representing a mapping from 1 to 1).*

Following the main principle outlined above and assuming that, without loss of generality, the currently considered node is labeled with variable x_i ($n \geq i \geq 1$) as well as the fact that all previously traversed nodes (i.e. all nodes labeled with variable x_l, where $n \geq l > i$) already establish the identity structure, the currently considered node can be transformed to the identity structure as follows:

Apply Toffoli gates such that all 1-paths are moved from the second to the first edge while, at the same time, all 1-paths are moved from the third edge to the fourth edge. To this end, determine the sets of 1-paths for each edge of the currently considered node (denoted by P_1, P_2, P_3, and P_4, respectively) as well as the corresponding sets of 0-paths of the currently considered node (i.e. paths that terminate in a 0-stub; denoted by $\overline{P}_1$, $\overline{P}_2$, $\overline{P}_3$, and $\overline{P}_4$, respectively). A path represents an input and, hence, contains a literal for each variable x_j with $1 \leq j < i$ which either occurs in positive phase (x_j) or negative phase ($\overline{x}_j$). Variable x_i is neglected, because it is inherently known ($\overline{x}_i$ for paths in P_1 and P_3, as well as x_i for paths in P_2 and P_4). Since the QMDD node describes a reversible function, the sets P_1 and P_3 are disjoint (both represent a mapping with input $x_i = 0$), i.e. $P_1 \cap P_3 = \emptyset$. Moreover, the set $\overline{P}_1$ of 0-paths through the first edge is equal to the set P_3 of 1-paths through the third edge. Finally, due to reversibility, the cardinalities of the sets P_2 and $\overline{P}_1 = P_3$ can be assumed to be equal.

Example 7. *Consider again the running example of Section III-A, i.e. the non-reversible function shown in Fig. 1a and one of its possible reversible embeddings as shown in Table Ib. The permutation matrix of this embedded function as well as the corresponding QMDD are both shown in Fig. 7. Consider the top node of this QMDD. The sets of 1-paths are $P_1 = \{\overline{x}_2\overline{x}_1, x_2\overline{x}_1\}$, $P_2 = \{\overline{x}_2 x_1, x_2\overline{x}_1\}$, $P_3 = \{\overline{x}_2 x_1, x_2 x_1\}$, and $P_4 = \{\overline{x}_2\overline{x}_1, x_2 x_1\}$. Note that $P_1 \cap P_3 = P_2 \cap P_4 = \emptyset$ and that $\overline{P}_1 = \{\overline{x}_2 x_1, x_2 x_1\} = P_3$ are the 0-paths through the first edge.*

Because of the relation between 1-paths and 0-paths discussed above, each 1-path of the second edge can be swapped with a 0-path of the first edge. To keep the number of required Toffoli gates as small as possible, we swap a 1-path $p \in P_2$



(a) Matrix (b) QMDD

Fig. 7: Representations for the embedded function (Table. Ib)

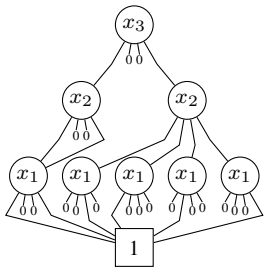


Fig. 8: QMDD after processing the top node

with its most similar 0-path $p' \in \overline{P}_1$. In fact, if there exist corresponding paths p and p' which are identical ($p = p'$), only a Toffoli gate with target line x_i and a set of control lines that represent p is required.[2] If $p \neq p'$, p has to be adjusted to match p' before the paths can be swapped as described above. To this end, a Toffoli gate with target line x_j is added for each variable x_j that occurs in p and p' in different phases. Note that all these Toffoli gates contain a positive control line x_i, since only the paths in the second and fourth edge shall be changed. Swapping all 1-paths of the second edge with the 0-paths of the first edge inherently swaps the 1-paths of the third edge with the 0-paths of the fourth edge and, hence, transforms the currently considered node to the identity structure.

Example 7 (continued)**.** *Consider again the top node of the QMDD depicted in Fig. 7b. As can be seen, there exists a 1-path $p \in P_2$ which is identical to a 0-path $p' \in \overline{P}_1$, namely $p = \overline{x}_2 x_1 = p'$. To swap the 1-path with the 0-path, a gate $TOF(\{\overline{x}_2, x_1\}, x_3)$ is applied. The remaining 1-path $q = x_2\overline{x}_1$ of P_2 has to be adjusted to match the 0-path $q' = x_2 x_1$. This requires a gate $TOF(\{x_3, x_2\}, x_1)$. Since afterwards $q = q'$, the paths can be swapped using a gate $TOF(\{x_2, x_1\}, x_3)$ – resulting in the QMDD shown in Fig. 8 where the currently considered node assumes the identity structure. Continuing these steps for all remaining nodes eventually yields the circuit shown in Fig. 9 realizing the function represented by the QMDD in Fig. 7b (and, hence, the function described in Fig. 1a in reversible logic).*

[2]As discussed above, the set of control lines is additionally enriched with literals that specify the path to the currently considered node.

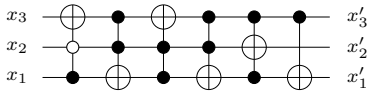


Fig. 9: Circuit obtained by the established design flow

IV. One-pass Design of Reversible Circuits

The established design flow reviewed in Section III requires two separate tasks to be addressed, namely embedding and synthesis. While, in recent years, impressive progress has been made for both steps with respect to efficiency and scalability (see e.g. [29], [40] for the embedding step and [28], [26] for the synthesis step), this approach still suffers from the need to conduct these steps separately. More precisely, this leads to the following main drawbacks:

- An embedding function is defined rather arbitrarily and without considering the following synthesis step. As an example, there are 192 possibilities how to assign precise values for the $*$- and the -entries of the function considered in Example 4. But eventually the embedding process just picks a single solution without considering which embedding might be particularly suited for the synthesis steps to be conducted afterwards. This way, a huge degree of freedom is not exploited (since all 16 *don't cares* are already assigned prior to synthesis).
- An embedding function almost always requires garbage outputs and, hence, constant inputs which, in turn, lead to an exponential increase in the truth table and/or function matrix description. As a consequence, the actual synthesis is performed on a function representation which is usually much more complex than the representation of the originally intended target function. This poses a threat to the efficiency and scalability of the synthesis process.

An alternative to address these drawbacks is proposed in this work. To this end, we introduce the concept of one-pass design of reversible circuits which, instead of considering embedding and the actual synthesis separately, conducts both tasks at once. In the following, this new design concept is introduced as follows: We first discuss the importance of the embedding process, i.e. we analyze why problems arise when functional synthesis approaches are applied without a prior embedding process[3]. Afterwards, two complementary solutions overcoming these problems and, hence, realizing one-pass design of reversible circuits are proposed: The first one (introduced in Section IV-B) guarantees the minimum with respect to the number of required circuit lines, but addresses only the first drawback (exploiting the full degree of freedom). The second one (introduced in Section IV-C) also addresses the second drawback (keeping the function representation small), but may require a slightly larger (but still bounded) number of additional circuit lines. Both solutions can be incorporated to various synthesis schemes proposed in the past such as e.g. [28], [26]. In the following, we are using the solution proposed in [28] and reviewed in Section III-B (and recently improved in [39]) as a representative. Experimental evaluations

[3]A preliminary study on that has recently been conducted in [41].

(summarized later in Section V) clearly show that the proposed one-pass design of reversible circuits clearly addresses the drawbacks discussed above.

A. Importance of the Embedding Process

In this section, we analyze why problems arise when functional synthesis approaches are applied without a prior embedding process. The corresponding findings are afterwards utilized to re-develop reversible circuit synthesis towards the proposed one-pass design flow.

Recall the synthesis approach reviewed in Section III-B as a representative. The main idea is to swap the 1-paths of the second and third edge with the 0-paths of the first and fourth edge, respectively. This only works if $\left|\overline{P}_1 \geq P_2\right|$ and $\left|\overline{P}_4 \geq P_3\right|$ for the currently considered node, i.e. there must be at least as many 0-paths through the first edge as 1-path through the second edge. This is always the case if the function to be synthesized is reversible, because each output pattern occurs exactly once. However, in case the function to be synthesized is non-reversible (because the embedding step has been skipped), QMDD nodes will occur for which the set of 0-paths through the first edge $\overline{P}_1$ contains fewer paths than the set of 1-paths through the second edge P_2, i.e. $\left|\overline{P}_1\right| < |P_2|$. Then, not all 1-paths of the second edge can be moved to the first edge, i.e. the identity structure cannot be established for the node. Consequently, the synthesis approach fails[4].

Example 8. *Consider the top node of the QMDD shown in Fig. 3 representing the function matrix of the non-reversible function shown in Fig. 1a. The respective set of paths are* $P_1 = \{\overline{x}_1, x_1\}$, $P_2 = \{x_1\}$, $P_3 = \emptyset$, *and* $P_4 = \{\overline{x}_1\}$. *Since* $\overline{P}_1 = \emptyset$ *does not contain any 0-paths that can be swapped with the path of* P_2, *the synthesis approach described in Section III-B cannot be applied. Consequently, there is no chance to transform the currently considered node to the identity structure and, hence, the synthesis algorithm fails.*

In the following sections, we describe how to overcome this problem. The approach proposed in Section IV-B solves this problem by inserting additional 0-paths into the QMDD. To this end, the minimum number of additionally required circuits lines is determined first (allowing for the proper insertion of the new paths and, eventually, leading to an embedding). Since the number of required circuit lines is guaranteed to be the minimum here, we denote this the *exact solution*. The approach proposed in Section IV-C modifies the function to be synthesized such that synthesis can continue without running into the problems discussed above. In order to later restore the modifications on the function, additional circuit lines are employed which buffer any changes to the function. This may require more circuit lines than the minimum number (although still bounded), which is why we denote this the *heuristic solution*. Both (complementary) schemes conduct embedding *during* the synthesis rather than prior to synthesis – leading to the desired one-pass design of reversible circuits.

[4]Note that similar problems arise when other synthesis approaches such as e.g. [26] are applied without embedding.

Fig. 10: QMDD representing the garbage variables

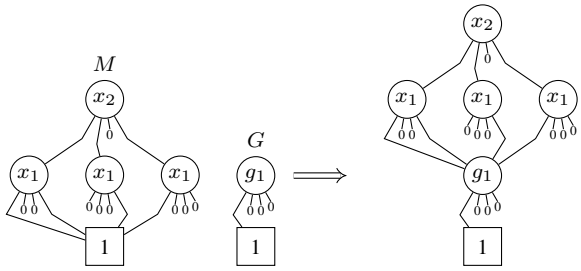


Fig. 11: Extending the function matrix

B. Exact Solution

To overcome the problem with the mismatching cardinalities of $\overline{P}_1$ and P_2 discussed above, we propose to increase the overall number of 0-paths in the QMDD *M* without increasing the number of 1-paths. To this end, we add $k' = (m+k) - \max(n,m)$ garbage variables to the QMDD prior to synthesis (if k' is not negative; otherwise no further variables have to be added)[5]. These garbage variables build up a QMDD *G* as illustrated in Fig. 10, which has a single 1-path, namely $p = \overline{a}_{k'} \ldots \overline{a}_2 \overline{a}_1$. The garbage variables are added to *M*, by replacing its terminal node with the root node of *G* (i.e. forming the Kronecker product $M \otimes G$). Consequently, the original function can be obtained if all ancillary inputs are set to 0 (similarly as discussed in Section III-A). For all other combinations of the ancillary inputs, the output is *don't care*. These *don't cares* are represented by 0-paths, since they do not have to be considered during synthesis.

Example 9. *The left-hand side of Fig. 11 shows the QMDD* M *representing the function matrix of the non-reversible function provided in Fig. 1b. Since* $n = m = 2$ *and* $k = 1$, $k' = 2 + 1 - 2 = 1$ *additional variable – named* g_1 *– is added to the QMDD. The extended QMDD is shown on the right-hand side of Fig. 11.*

Adding variables to the QMDD leads to an incompletely specified function, for which a synthesis scheme similar to the one reviewed in Section III-B can be applied[6]. However,

[5]Note that k can be determined efficiently as described in [29] and [40].

[6]Note that there exist approaches for synthesis of incompletely specified functions (see e.g. [18], [10]) – although applicable for rather small functions only. Besides that, the assumption is applied there that the respectively given function can be made reversible by properly assigning the *don't cares* only. The methodology proposed in this work is applicable for arbitrary functions and does not rely on such an assumption.

the QMDD contains $2^{k'}$ times fewer 1-paths (and, hence, contains more 0-paths), i.e. less paths have to be considered in order to accomplish the identity structure compared to the established two-stage design flow. Additionally, since the number of 0-paths $p' \in \overline{P}_1$ is larger than the number of 1-paths $p \in P_2$ for most nodes (i.e. $\left|\overline{P}_1\right| > |P_2|$) some degree of freedom is introduced. In fact, this is the same degree of freedom which is available when conducting the embedding as described in Section III-A. However, in contrast to the established design flow, this degree of freedom can now be exploited during synthesis, because the embedding that suits best (in order to reduce the complexity of the circuit) is implicitly chosen.

More precisely, the degree of freedom allows for choosing the subset of $\overline{P}_1$ containing the 0-paths that require the fewest number of Toffoli gates when swapped with the 1-paths of P_2. Note that we have to consider the 1-paths through the third edge separately, because swapping all 1-paths through the second edge with 0-paths through the first edge does not necessarily swap all 1-paths through the third edge with 0-paths through the fourth edge. In the case that $\left|\overline{P}_1\right| = |P_2|$ for the currently considered node, synthesis is conducted as described in Section III-B.

Once the second and third edge are transformed to 0-stubs, the currently considered QMDD node does not necessarily have the desired identity structure – the first or the fourth edge may be a 0-stub as well. In this case, we insert an edge to accomplish the identity structure (exploiting the degree of freedom that some one-to-one mappings can arbitrarily be defined). In other words, inserting an edge is nothing but adding missing 1-paths in a way that best suits deriving the desired identity structure.

Example 10. *Consider the top node of the QMDD shown in Fig. 11 (representing the incomplete function resulting from extending the QMDD with one additional variable), which contains only four 1-paths. The respective set of paths are* $P_1 = \{\overline{x}_1\overline{g}_1, x_1\overline{g}_1\}$, $P_2 = \{x_1\overline{g}_1\}$, $P_3 = \emptyset$, *and* $P_4 = \{\overline{x}_1\overline{g}_1\}$. *The degree of freedom allows for arbitrarily choosing 0-paths from* $\overline{P}_1 = \{\overline{x}_1 g_1, x_1 g_1\}$ *that should be swapped with the 1-paths* $p \in P_2$. *There is only one path* $p = x_1\overline{g} \in P_2$. *Unfortunately, this path is not contained in* $\overline{P}_1$. *Therefore, we chose the most similar path* $p' \in \overline{P}_1$, *which is* $p' = x_1 g_1$. *To adjust the paths, we apply gate* $TOF(\{x_2, x_1\}, g_1)$, *and eventually gate* $TOF(\{x_1, g_1\}, x_2)$ *to swap them. The resulting QMDD is shown on the left-hand side of Fig. 12.*

Now, this QMDD already realizes the desired identity structure for almost all nodes labeled with x_2 *and* x_1. *Only the fourth edge of the right node labeled* x_1 *is a 0-stub (although we require a 1-path here). However, this can be addressed without the need to add any further gates. In fact, the degree of freedom allows us to simply add a new 1-path through this edge as shown at the right-hand side of Fig. 12. Note that, if embedding would have been conducted prior to synthesis, 1-paths would have been set very likely in a fashion which require additional gates. In contrast, the one-pass scheme proposed here allows to set these 1-paths appropriately without leading to further costs.*

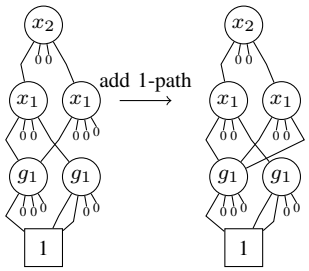


Fig. 12: Incompletely specified

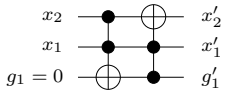


Fig. 13: Circuit obtained by the exact one-pass design

The synthesis scheme outlined above only needs to be conducted for QMDD nodes representing primary variables. Any additional variables (e.g. g_1 in the example) can be mapped arbitrarily as long as the dependencies discussed in Section III-A are considered. This, however, is implicitly the case since only reversible gates are applied thus far – eventually realizing a fully reversible function. Again, this fully exploits the degree of freedom as a prior embedding step may have realized a mapping of additional variables such as g_1 which would require further gates.

Example 10 (continued)**.** *Considering the current QMDD as shown at the right-hand side of Fig. 12, it can be seen that all nodes labeled by primary variables already realize the desired identity structure. In contrast, the mapping of the additional variable* g_1 *can arbitrarily be realized (reversibility is guaranteed by the fact that only reversible gates have been employed thus far). Hence, we can terminate the synthesis process and obtain the resulting circuit as shown in Fig. 13.*

C. Heuristic Solution

The exact solution proposed in the previous section yields circuits with the minimum number of circuit lines. Even though experimental results (summarized later in Section V) demonstrate the scalability of this approach, one of the drawbacks remains: the QMDD has to be extended by additionally required variables prior to synthesis – causing an exponential overhead in the worst case (as discussed above). In order to overcome this drawback, an alternative solution is proposed in this section that may require more (although still a bounded number of) additional circuit lines, but instead does not lead to an increase of the function representation.

The main idea is to stay with the originally given function representation (i.e. a QMDD representing a function matrix without additional variables) and to modify this (non-reversible) function to be synthesized whenever a QMDD node is encountered for which the second edge has more 1-paths than the first edge has 0-paths (the problem discussed in Section IV-A). Since this obviously leads to a circuit that realizes a function different to the desired one, these

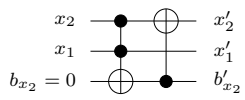


Fig. 14: Circuit obtained by the heuristic one-pass design

modifications are stored on so-called buffer lines and reverted after synthesis terminated.

More precisely, a QMDD node with $\left|\overline{P}_1\right| \geq |P_2|$ and $\left|\overline{P}_4\right| \geq |P_3|$ is transformed to the identity structure as described above[7]. In case the second edge has more 1-paths than the first edge has 0-paths (i.e. $\left|\overline{P}_1\right| < |P_2|$) not all 1-paths through the second edge can be moved to the first edge. Then, we swap as many 1-paths through the second edge with 0-paths through the first edge and apply the respective gates. Afterwards, we swap the remaining 1-paths $p \in P_2$ with 0-paths $p' \in \overline{P}_4$ to establish the identity structure. This can be conducted by flipping the output of the currently considered variable x_i for these paths (since $P_2 \subseteq \overline{P}_4$). Obviously, this changes the function to be synthesized because we flip bit i in the output pattern for some inputs. To remember these bit flips on bit i, we add a so-called buffer-line b_i – initialized with zero – onto which we store all input combinations which are affected by this flip. These input combinations can be derived from the respectively considered path, i.e. we apply a Toffoli gate $TOF(\{p\}, b_i)$ for each 1-path $p \in P_2$ (the set of control lines is enriched by literals that represent the path to the currently considered node as well as with literal x_i). An example illustrates the idea:

Example 11. *Consider the top node of the QMDD shown in Fig. 3 representing the (non-reversible) function matrix from Fig. 1b. The respective set of paths are $P_1 = \{\overline{x}_1, x_1\}$, $P_2 = \{x_1\}$, $P_3 = \emptyset$, and $P_4 = \{\overline{x}_1\}$. Since $\overline{P}_1 = \emptyset$ does not contain any 0-path that can be swapped with a path from P_2, the synthesis approach described in Section III-B cannot be applied. Therefore, we have to modify the function to swap the 1-path of P_2 with the 0-path of $\overline{P}_4$ to obtain the desired identity structure. To this end, we have to flip the output x_2 for the path x_2x_1, resulting in the identity QMDD. Since this modification has to be restored after synthesis is completed, we store it on a buffer-line b_{x_2} (initialized with zero) by applying Toffoli gate $TOF(\{x_2, x_1\}, b_{x_2})$ (the first gate in Fig. 14).*

After synthesis has terminated, the buffer lines can be used to reverted the made modifications. To this end, for each buffer line b_i a single gate $TOF(\{b_{x_i}\}, x_i)$ is applied, which flips the output back to its intended value.

Example 11 (continued)**.** *Since the modification of the function described above already yields the identity QMDD, synthesis completes without adding further gates. Finally, the modification made on output x_2 has to be restored by applying gate $TOF(\{b_{x_2}\}, x_2)$, eventually resulting in the circuit shown in Fig. 14.*

[7]If the number of outputs is larger than the number of inputs, the function matrix describes an incompletely specified function, since it contains column consisting of 0-entries only.

TABLE II: Characteristics of design flows

	Established (Two-stage)	Proposed (One-pass) exact	heur.
Exploits full degree of freedom?	✗	✓	✓
No. variables for function representation	$\max(n, m+k)$	$\max(n, m+k)$	$\max(n, m)$
Yields circuit with min. lines?	✓	✓	✗

Although additional circuit lines have to be added dynamically during synthesis, their number is bounded by m, since in worst case each of the m output bits has to be flipped for at least one input combination. Therefore, at most m buffer lines (one for each output bit) are required – yielding a circuits with $n+m$ lines at maximum. Moreover, if the function to be synthesized is reversible, no additional lines are added during synthesis, because the cardinalities of the set of 1-paths and 0-paths always match.

Overall, both approaches presented above realize the proposed one-pass design flow and conduct both, the embedding step and the synthesis step, at the same time. By this, they fully exploit the degree of freedom as corresponding paths are set so that they perfectly suit the synthesis (ideally, the paths are set so that the desired identity structure is realized with no or significantly fewer gates). Moreover, the heuristic scheme does not even require additional variables for the respective function description and, by this, simplifies synthesis further (at the expense of not guaranteeing a circuit with the minimum number of lines). Table II summarizes the characteristics of the established design flow, as well of the proposed solutions for one-pass design of reversible circuits. As can be seen, the drawbacks of the established design flow discussed in the beginning of this section are addressed by the proposed solutions. Experimental evaluations summarized in the next section confirm the benefits of the new design flow.

V. Experimental Results

In order to evaluate the proposed one-pass design flow, we implemented both, the exact solution as well as the heuristic solution in C++ on top of the QMDD package [21], the BDD package CUDD [30], and RevKit [27].[8] As benchmarks, we used the Boolean functions available at RevLib [33]. In the following, the results of two series of evaluations are summarized: First, we compare the results obtained by the one-pass design to the best known synthesis approaches. Afterwards, a direct comparison between the existing two-stage design flow and the one-pass design flow is conducted using the same synthesis approach (namely QMDD-based synthesis reviewed in Section III-B) as basis. This allows for a more detailed evaluation of the improvements achieved by the newly proposed design flow (independently from the actually applied synthesis approach). All evaluations have been conducted on a machine with 32 GB of memory running Linux 4.4.

[8]An implementation of the proposed solutions is available at http://www.jku.at/iic/eda/one_pass_design_of_reversible_circuits.

A. Comparison to Previous Approaches

In a first evaluation, we aim for comparing the proposed one-pass design flow to the best available synthesis approaches for reversible circuits. To this end, we first consider the two-stage design flow employed by the embedding solution from [29] and the synthesis solution from [26]. They represent the best known functional synthesis solutions and are both publicly available in RevKit [27] (executable using the command *embed -b; tbs -b*), a toolkit that contains various methods for synthesis and optimization of reversible circuits and, hence, became a commonly used tool for the design of reversible circuits.

Table III provides the obtained results. The first four columns list the name of the benchmark, the number of inputs (n) and outputs (m), as well as the minimum number of required circuit lines (denoted by $min.$). The remaining columns list the runtime (in CPU seconds; denoted by *t*) as well as the costs (in terms of *T-depth*) of the circuits obtained by the two-stage approach (i.e. the RevKit implementations of [29], [26]) as well as by the exact and heuristic one-pass solutions proposed in Section IV-B and Section IV-C, respectively. The established two-stage design flow (composed of [29] and [26]) as well as the proposed exact one-pass solution yield circuits where the number of circuit lines is the minimum. Since this is not necessarily the case for the heuristic approach, Table III lists the obtained number of circuit lines for this scheme as well (denoted by l).

As can be clearly seen, the proposed one-pass scheme (in its exact realization) is faster in terms of runtime by magnitudes than the currently best known approach using a two-stage design flow. For all benchmarks, the minimum number of circuit lines could be determined within a fraction of a second. For all benchmarks that could be synthesized using the two-stage design flow (composed of [29] and [26]), the exact solution proposed in this paper requires significantly less runtime. Furthermore, in 21 out of the 26 benchmarks, for which the two-stage approach ran into a timeout of 10 000 seconds, the exact solution was able to generate a circuit (within a few seconds in most cases). For example, benchmarks *cm150a* and *mux* could be synthesized in less than three seconds, whereas the previously proposed method already required hundreds of seconds alone for embedding (and running into a timeout in the synthesis step). Furthermore, benchmark *frg1* could be synthesized in roughly 30 seconds, whereas the two-stage approach (composed of [29] and [26]) already timed out in the embedding phase.

The heuristic approach even allows to further improve with respect to runtime. In fact, using the heuristic one-pass synthesis, we were able to synthesize all benchmarks in reasonable time (including the ones which timed-out before). Although the number of circuit lines of the resulting circuits is not necessarily the minimum, the experiments show that a circuit with the minimum number of lines was generated in 40 out of 69 cases (highlighted bold in column l of Table III). Only in 29 cases, a slightly larger number of additional circuit lines is needed – however, never more than four more lines are needed. In contrast, the heuristic approach exploited the reduced complexity and generated all results in significantly less run-time. For the benchmarks *cordic*, *cps*, *apex2*, and *e64* a circuit with the minimum number of lines could be determined within 50 seconds, whereas the exact approach timed out. As discussed above, this was possible because the heuristic solution does not increase the function representation by additional variables.

Besides these runtime improvements, also a significant improvement in terms of *T-depth* can be observed compared to the previously proposed method. More precisely, improvements of 31% and 81% are reported on average for the solutions following the new one-pass design flow and proposed in Section IV-B and Section IV-C, respectively. However, also some cases exist where the costs increase (e.g. *0410184* and *ex5p*). This can be explained by the fact that both flows still rely on different synthesis schemes ([26] employs a so-called transformation-based synthesis, while the solutions proposed in Section IV-B and Section IV-C follows QMDD-based synthesis). Hence, a direct comparison between the existing two-stage design flow and the one-pass design flow using the same synthesis approach as basis is desirable (and is covered in the next section).

Before that, however, we briefly discuss how the obtained results stand to structural synthesis approaches [9], [38], [31], [25]. Such methods yield circuits with significantly more lines than the minimum (e.g. magnitudes above the minimum for [25] and $2n + m$ for [9]). This is disadvantageous, since each line has to be represented physically in the underlying system (e.g. by a qubit in quantum computation). As already mentioned in Section I, this is why we focused on synthesis approaches in this work, which keep the number of lines as close as possible to the minimum. Since additionally the huge differences in the number of lines between functional approaches as considered in this work and structural approaches as considered e.g. in [9], [38], [31], [25] also affects the resulting costs of the circuits (as already evaluated in [37]), a numerical comparison would be misleading.

B. Comparison of the Design Flows

In this section, we compare the proposed one-pass design flow to the established design flow using the same underlying synthesis approach. To this end, we again followed the established two-stage design flow, but now used the QMDD-based synthesis approach (originally introduced in [28] and recently improved in [39]) reviewed in Section III-B instead of the synthesis approach proposed in [26]. This allows to evaluate the benefits of the proposed one-pass design methodology compared to the established two-stage approach without any side-effects caused by the utilization of different synthesis approaches as it was the case in the evaluations summarized in the previous section. More precisely, we again utilize the embedding provided by RevKit[9] and pass the resulting function to the QMDD-based synthesis approach. Afterwards, we compared the obtained results with the circuits generated by the proposed one-pass design solutions (which rely on

[9]Obtained from the circuits resulting from the approach discussed above.

TABLE III: Comparison to the best functional approach of RevKit

				Best known (two-stage) solution ([29], [26])			Proposed one-pass design					
							Exact solution (Sec. IV-B)			Heuristic solution (Sec. IV-C)		
Benchmark	n	m	$min.$	t_{emb}	t_{synth}	T-depth	t_k	t_{synth}	T-depth	t_{synth}	l	T-depth
sqrt8	8	4	9	0.01	0.48	34 704	0.00	0.02	7 410	0.01	12	3 720
rd84	8	4	11	0.01	0.88	39 756	0.00	0.03	54 150	0.01	**11**	16 551
adr4	8	5	9	0.00	0.16	23 781	0.00	0.02	57 870	0.01	**9**	4 452
radd	8	5	9	0.00	0.66	35 067	0.00	0.02	44 088	0.02	**9**	15 036
dist	8	5	10	0.00	0.70	37 449	0.00	0.02	70 329	0.03	13	13 584
root	8	5	10	0.00	0.72	36 981	0.00	0.02	18 036	0.02	13	8 928
dc2	8	7	13	0.00	1.45	48 837	0.00	0.03	48 843	0.02	14	4 290
misex1	8	7	14	0.00	1.98	65 316	0.00	0.02	30 201	0.02	15	2 106
hwb8	8	8	8	0.01	0.49	31 416	0.00	0.03	62 610	0.07	**8**	60 774
urf2	8	8	8	0.01	0.38	31 257	0.00	0.05	52 803	0.10	**8**	51 207
mlp4	8	8	13	0.02	0.95	48 141	0.00	0.11	132 132	0.02	16	26 466
ex5p	8	63	68	0.07	153.47	146 901	0.00	6.08	3 477 078	2.49	70	763 383
9symml	9	1	10	0.01	1.88	99 381	0.00	0.02	44 856	0.01	**10**	17 184
life	9	1	10	0.00	2.94	100 227	0.00	0.01	44 520	0.02	**10**	11 196
max46	9	1	10	0.01	3.01	98 289	0.00	0.01	29 304	0.02	**10**	10 248
sym9	9	1	10	0.00	2.17	999 381	0.00	0.02	44 856	0.02	**10**	17 184
clip	9	5	11	0.02	3.59	102 489	0.00	0.08	153 597	0.04	14	57 108
hwb9	9	9	9	0.02	2.27	88 452	0.00	0.06	184 998	0.21	**9**	180 426
urf1	9	9	9	0.01	1.65	81 399	0.01	0.10	146 208	0.14	**9**	144 132
urf5	9	9	9	0.02	0.46	38 898	0.00	0.04	50 259	0.05	**9**	49 659
dk27	9	9	15	0.01	6.59	123 276	0.00	0.04	118 404	0.04	18	3 333
apex4	9	19	26	0.07	43.33	170 100	0.00	1.28	1 618 155	2.48	28	608 064
sym10	10	1	11	0.02	9.46	238 674	0.00	0.03	75 936	0.02	**11**	28 656
sao2	10	4	14	0.01	26.26	296 841	0.00	0.12	334 245	0.02	**14**	28 200
alu2	10	6	14	0.03	20.22	308 214	0.00	0.16	492 450	0.07	16	60 231
example2	10	6	14	0.03	31.33	308 214	0.00	0.12	492 450	0.06	16	60 231
x2	10	7	16	0.01	43.14	391 404	0.00	0.09	217 707	0.03	17	10 218
alu3	10	8	14	0.02	25.17	337 281	0.00	0.12	399 315	5.42	18	64 728
urf3	10	10	10	0.04	7.03	207 186	0.01	0.20	326 532	0.37	**10**	321 684
ex1010	10	10	18	0.07	97.61	475 536	0.01	0.47	1 665 222	0.01	20	175 881
dk17	10	11	19	0.01	103.51	492 033	0.00	0.32	1 679 478	0.08	21	42 948
apla	10	12	22	0.02	202.47	604 542	0.00	0.52	1 643 904	0.15	**22**	61 599
cm152a	11	1	11	0.04	45.37	638 454	0.00	0.02	528	0.02	12	384
cm85a	11	3	13	0.01	55.49	674 883	0.00	0.03	69 360	0.04	14	15 972
urf4	11	11	11	0.11	65.39	637 539	0.02	0.74	1 231 530	1.87	**11**	1 201 722
add6	12	7	13	0.10	326.96	1 606 533	0.03	0.58	1 580 208	0.11	**13**	468 672
alu1	12	8	18	0.08	1206.68	2 389 212	0.00	0.35	1 308 360	0.05	20	90 051
Cycle10_2	12	12	12	0.14	0.01	1 926	0.00	0.06	2 979	0.11	**12**	2 979
plus63mod4096	12	12	12	0.11	0.69	47 694	0.00	0.08	2 526	0.06	**12**	918
plus127mod8192	13	13	13	0.25	3.40	115 833	0.00	0.15	3 246	0.12	**13**	1 098
plus63mod8192	13	13	13	0.26	1.08	70 698	0.00	0.15	3 246	0.12	**13**	1 146
co14	14	1	15	0.18	8844.44	9 399 120	0.00	0.01	26 520	0.02	**15**	16 260
alu4	14	8	19	0.43	>10000.00		0.01	13.78	18 523 833	0.01	22	3 057 696
f51m	14	8	19	0.97	>10000.00		0.00	6.48	12 171 363	5.35	22	5 354 592
tial	14	8	19	0.55	>10000.00		0.02	15.42	18 416 949	4.15	22	3 028 788
cu	14	11	25	0.24	>10000.00		0.00	1.29	5 084 367	0.10	**25**	36 513
0410184	14	14	14	0.42	0.06	11 256	0.00	0.30	122 748	0.30	**14**	122 748
misex3	14	14	28	1.24	>10000.00		0.01	44.16	57 810 417	1.69	**28**	1 212 048
misex3c	14	14	28	1.21	>10000.00		0.00	41.16	55 638 522	1.43	**28**	1 306 836
table3	14	14	28	0.67	>10000.00		0.00	66.86	60 134 850	0.52	**28**	342 591
In0	15	11	25	0.41	>10000.00		0.00	83.95	82 366 977	2.43	26	1 572 276
ham15	15	15	15	3.00	>10000.00		0.06	2.00	19 864 608	4.32	**15**	17 998 662
urf6	15	15	15	1.67	>10000.00		0.00	1.29	2 082 000	1.40	**15**	2 081 760
ryy6	16	1	17	1.21	>10000.00		0.00	0.19	465 300	0.02	**17**	20 532
t481	16	1	17	1.73	>10000.00		0.00	2.70	3 319 272	0.11	**17**	247 752
cmb	16	4	20	1.30	>10000.00		0.00	0.17	817 131	0.05	**20**	5 895
pcler8	16	5	21	1.25	>10000.00		0.01	0.65	1 941 297	0.06	**21**	17 451
cm163a	16	13	25	0.34	>10000.00		0.02	8.48	21 546 000	0.12	27	50 760
pdc	16	40	55	1.23	>10000.00		0.01	2343.61	457 221 879	11.93	56	2 836 584
spla	16	46	61	1.84	>10000.00		0.00	568.87	367 270 986	7.02	62	4 214 007
cm151a	19	9	27	5.81	>10000.00		0.00	3272.97	514 176 591	0.27	28	46 314
cm150a	21	1	22	682.41	>10000.00		0.00	2.62	252 978	0.72	**22**	1 200
mux	21	1	22	735.04	>10000.00		0.00	2.53	289 722	0.38	**22**	1 128
cordic	23	2	25	870.74	>10000.00		0.00	>10000.00		3.83	**25**	10 517 292
cps	24	109	132	500.94	>10000.00		0.02	>10000.00		20.99	**132**	6 253 920
frg1	28	3	30	>10000.00	>10000.00		0.00	30.35	26 147 451	0.27	31	118 095
apex2	39	3	42	>10000.00	>10000.00		0.32	>10000.00		5.41	**42**	2 404 404
seq	41	35	75	>10000.00	>10000.00		0.00	>10000.00		241.73	76	39 591 105
e64	65	65	129	>10000.00	>10000.00		0.01	>10000.00		46.42	**129**	24 515 583

n: primary inputs m: primary outputs $min.$: minimum number of required lines t_{emb}: time required for embedding
t_{synth}: time required for synthesis t_k: time required to determine k l: number of lines of the resulting circuit

QMDD-based synthesis as well). This way, a fair comparison between both synthesis flows can be conducted.

The results are presented in Table IV. The first four columns of the table again list the name of the benchmark, the number of primary inputs (n) and primary outputs (m), as well as the minimum number of required lines ($min.$). The remaining columns list the time required for synthesis (columns labeled t) and the *T-depth* of the resulting circuit for the two-stage design flow as well as for the proposed one-pass solutions. For the heuristic solution we also list the number of lines of the resulting circuits (l). Again, the cases where this number of lines is the actual minimum are highlighted in bold. Note that we omitted benchmarks which already represented a reversible function since, in these cases, all three approaches yield the same circuit. Besides that, we only considered benchmarks for which the two-stage design flow was able to derive an embedding within the given timeout (that the proposed one-pass design flow can handle many of these cases has already been shown by means of Table III).

First, we compare the exact solution (proposed in Section IV-B) to the conventional two-stage approach. For all benchmarks, we observe a significant improvement in terms of *T-depth* – leading to an improvement of 96% on average. Since the underlying synthesis algorithm of both approaches is equal (namely QMDD-based synthesis), this improvement can completely be attributed to the fact that one-pass synthesis and, hence, the discussed degree of freedom is exploited.

Next, we compare the heuristic solution to the exact solution. The heuristic approach is much faster than the exact one (as discussed above), since the QMDD is not enriched by further variables prior to synthesis (in contrast to the two-stage design flow and the exact design flow which are extended to a total of $\max(n, m + k)$ variables, the heuristic solution stays with the originally given $\max(n, m)$ variables). This allows for a more compact representation of the function to be synthesized. Hence, also the second benefit discussed above can fully be exploited here. Besides that, this even yields further improvements with respect to the *T-depth*, since the more variables the function to be synthesized has, the more likely it is that a gate has a larger number of control lines (which causes higher costs). In fact, another reduction of 80% (78% if we consider only those benchmarks for which the resulting circuit has a minimum number of lines) compared to the exact solution can be observed – resulting in an average improvement by a factor of 185 compared to original the two-stage design flow. We expect similar improvements when applying the proposed one-pass design flow to other functional synthesis approaches.

Overall, the experimental evaluation confirms the benefits of the proposed one-pass design flow (which can be also applied to many functional synthesis approaches) over the conventional two-stage design flow. Besides substantial speedups compared to the state of the art design flow (for some benchmarks a reversible circuit with the minimum number of lines has been synthesized for the first time), substantial improvements in terms of *T-depth* were observed. In the direct comparison using the same synthesis approach as basis, costs could have been reduced by a factor of 185 on average. Hence, one-pass design clearly outperforms the currently established functional design flow for reversible circuits where embedding and synthesis are conducted separately.

VI. Conclusion

In this work, we proposed a novel design flow for functional synthesis of reversible circuits, which can be applied to many functional synthesis approaches. In contrast to the conventional two-stage design flow, which is composed of an embedding and a synthesis step, the proposed one-pass design flow combines both processes. This way, the full degree of freedom is exploited during synthesis which, additionally, can be conducted without a (worst case) exponential increase of the function representation. In order to evaluate these benefits, an exact as well as a heuristic solution of the one-pass design scheme have been proposed and compared to the best known solution for functional synthesis of reversible circuits. The results showed significant improvements by orders of magnitudes with respect to runtime as well as circuit costs. For some benchmarks, a reversible circuit with the minimum number of lines has been synthesized for the first time. Hence, a completely new design scheme for reversible circuits has been introduced which clearly outperforms the previously established design flow. As an idea for future work, it would be interesting to see, how the one-pass design flow proposed in this work affects other synthesis approaches, which have been proposed by the community in the past.

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] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[3] W. C. Athas and L. J. Svensson. Reversible logic issues in adiabatic cmos. In *Workshop on Physics and Computation*, pages 111–118, Nov 1994.

[4] 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.

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

[6] 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.

[7] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[8] R. Drechsler and R. Wille. From truth tables to programming languages: Progress in the design of reversible circuits. In *International Symposium on Multiple-Valued Logic, ISMVL*, pages 78–85, 2011.

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

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

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

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

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

TABLE IV: Comparison of the design flows using QMDD-based synthesis

				Two-stage design flow (based on QMDD-based synthesis, Sec. III-B)			One-pass design flow (based on QMDD-based synthesis) Exact solution (Sec. IV-B)		Heuristic solution (Sec. IV-C)		
Benchmark	n	m	$min.$	t_{emb}	t_{synth}	T-depth	t_{synth}	T-depth	t_{synth}	l	T-depth
sqrt8	8	4	9	0.01	0.35	165 786	0.02	7 410	0.01	12	3 720
rd84	8	4	11	0.01	1.81	1 191 150	0.03	54 150	0.01	**11**	16 551
adr4	8	5	9	0.00	0.20	161 343	0.02	57 870	0.01	**9**	4 452
radd	8	5	9	0.00	0.22	162 426	0.02	44 088	0.02	**9**	15 036
dist	8	5	10	0.00	0.71	456 318	0.02	70 329	0.03	13	13 584
root	8	5	10	0.00	0.57	431 706	0.02	18 036	0.02	13	8 928
dc2	8	7	13	0.00	16.65	7 406 802	0.03	48 843	0.02	14	4 290
misex1	8	7	14	0.00	47.42	17 327 910	0.02	30 201	0.02	15	2 106
mlp4	8	8	13	0.02	12.14	7 206 060	0.11	132 132	0.02	16	26 466
9symml	9	1	10	0.01	1.04	462 042	0.02	44 856	0.01	10	17 184
life	9	1	10	0.00	0.76	479 166	0.01	44 520	0.02	**10**	11 196
max46	9	1	10	0.01	0.87	461 919	0.01	29 304	0.02	**10**	10 248
sym9	9	1	10	0.00	0.80	462 042	0.02	44 856	0.02	**10**	17 184
clip	9	5	11	0.02	2.92	1 192 170	0.08	153 597	0.04	14	57 108
dk27	9	9	15	0.01	317.93	39 853 860	0.04	118 404	0.04	18	3 333
sym10	10	1	11	0.02	3.15	1 210 083	0.03	75 936	0.02	**11**	28 656
sao2	10	4	14	0.01	58.91	17 137 734	0.12	334 245	0.02	**14**	28 200
alu2	10	6	14	0.03	94.42	17 850 822	0.16	492 450	0.07	16	60 231
example2	10	6	14	0.03	68.82	17 850 822	0.12	492 450	0.06	16	60 231
x2	10	7	16	0.01	3608.91	97 229 793	0.09	217 707	0.03	17	10 218
alu3	10	8	14	0.02	90.52	17 895 966	0.12	399 315	5.42	18	64 728
apla	10	12	22	0.02	>10000.00		0.52	1 643 904	0.15	**22**	61 599
cm152a	11	1	11	0.04	5.28	1 153 284	0.02	528	0.02	12	384
cm85a	11	3	13	0.01	36.52	7 477 437	0.03	69 360	0.04	14	15 972
add6	12	7	13	0.10	59.37	7 324 170	0.61	1 580 208	0.11	**13**	468 672
co14	14	1	15	0.18	968.70	21 061 593	0.01	26 520	0.02	**15**	16 260

n: primary inputs m: primary outputs $min.$: minimum number of required lines t_{emb}: time required for embedding
t_{synth}: time required for synthesis l: number of lines of the resulting circuit

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

[15] 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.

[16] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *Int'l Symp. on Multi-Valued Logic*, page 6, 2006.

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

[18] M. Mohammadi and M. Eshghi. Heuristic methods to use don't cares in automated design of reversible and quantum logic circuits. *Quantum Information Processing*, 7(4):175–192, 2008.

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

[20] P. Niemann, S. Basu, A. Chakrabarti, N. K. Jha, and R. Wille. Synthesis of quantum circuits for dedicated physical machine descriptions. In *Int'l. Conf. on Reversible Computation*, pages 248–264, 2015.

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

[22] A. Rauchenecker, T. Ostermann, and R. Wille. Exploiting reversible logic design for implementing adiabatic circuits. *In Int'l Conf. on Mixed Design of Integrated Circuits and Systems*.

[23] M. Saeedi and I. L. Markov. Synthesis and optimization of reversible circuits - a survey. *ACM Comput. Surv.*, 45(2):21, 2013.

[24] 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.

[25] M. Soeken and A. Chattopadhyay. Unlocking efficiency and scalability of reversible logic synthesis using conventional logic synthesis. In *Design Automation Conf.*, pages 149:1–149:6, 2016.

[26] M. Soeken, G. W. Dueck, and D. M. Miller. A fast symbolic transformation based algorithm for reversible logic synthesis. In *Int'l. Conf. on Reversible Computation*, pages 307–321, 2016.

[27] 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.

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

[29] M. Soeken, R. Wille, O. Keszocze, D. M. Miller, and R. Drechsler. Embedding of large Boolean functions for reversible logic. *J. Emerg. Technol. Comput. Syst.*, 12(4):41:1–41:26, Dec. 2015.

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

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

[32] R. Wille, R. Drechsler, C. Osewold, and A. G. Ortiz. Automatic design of low-power encoders using reversible circuit synthesis. In *Design, Automation and Test in Europe*, pages 1036–1041, 2012.

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

[34] 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.

[35] 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*, 2016.

[36] R. Wille, M. Soeken, and R. Drechsler. Reducing the number of lines in reversible circuits. In *Design Automation Conf.*, pages 647–652, 2010.

[37] 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.

[38] 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.

[39] A. Zulehner and R. Wille. Improving synthesis of reversible circuits: Exploiting redundancies in paths and nodes of QMDDs. In *Int'l. Conf. on Reversible Computation*, 2017.

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

[41] A. Zulehner and R. Wille. Skipping embedding in the design of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, 2017.

[42] 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.