RevVis: Visualization of Structures and Properties in Reversible Circuits

Robert Wille[1,2], Jannis Stoppe[2],
Eleonora Schönborn[1], Kamalika Datta[3], and Rolf Drechsler[1,2]

[1] Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2] Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3] Department of Information Technology, Bengal Engineering, Shibpur, India
{rwille,jstoppe,eleonora,drechsle}@informatik.uni-bremen.de
kdatta.iitkgp@gmail.com
www.informatik.uni-bremen.de/agra/eng/revvis.php

Abstract. The recent interest in reversible computation led to plenty of (automatic) approaches for the design of the corresponding circuits. While this automation is desired in order to provide a proper support for the design of complex functionality, often a manual consideration and human intuition enable improvements or provide new ideas for design solutions. However, this manual interaction requires a good understanding of the structure or the properties of a reversible cascade which, with increasing circuit size, becomes harder to grasp. Visualization techniques that abstract irrelevant details and focus on intuitively displaying important structures or properties provide a solution to this problem and have already successfully been applied in other domains such as design of conventional software, hardware debugging, or Boolean satisfiability. In this work, we introduce *RevVis*, a graphical interface which visualizes structures and properties of reversible circuits. *RevVis* collects relevant data of a given reversible cascade and presents it in a simple but intuitive fashion. By this, *RevVis* unveils information on characteristic structures and properties of reversible circuits that could be utilized for further optimization. A case study demonstrates this by considering circuits obtained from several synthesis approaches.

1 Introduction

Motivated by applications e.g. in quantum computation [1], low-power design [2], or encoder and decoder design [3], research in reversible computation received significant interest in the past. While rather small circuits have (manually) been considered at the beginning, a recent strive for automated and scalable methods supporting the design of several thousand gate circuits can be observed today. This resulted in plenty of (automated) approaches for the synthesis and optimization of reversible circuits in the recent past (see e.g. [4, 5] for overviews). Most of them are based on a particular genuine idea, e.g. reversible transformations at a truth-table description, the utilization of proper data structures such as ESOPs, decision diagrams, etc., or the application of templates. But besides that, also human intuition often led to ideas for new strategies to be exploited or enabled further improvements which could not be detected by a machine.

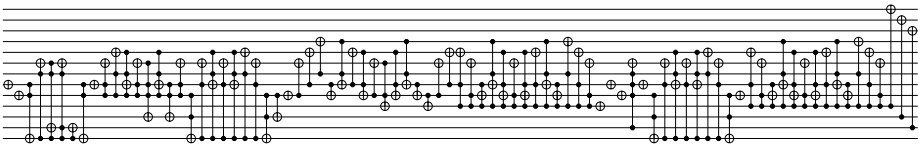


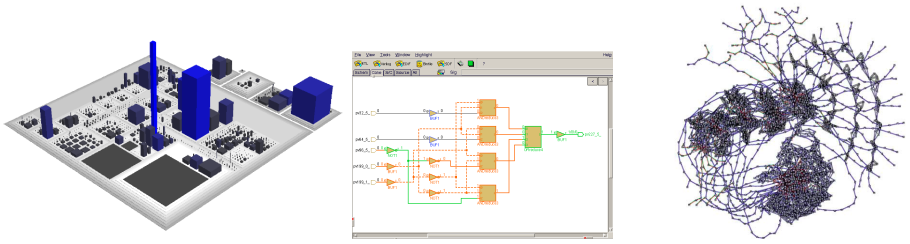
Fig. 1. Existing netlist visualization of reversible circuits

However, getting a good intuition of a considered circuit requires a deep technical understanding of how design approaches actually realize the respective circuits. Moreover, these approaches may generate circuits with certain structures and properties that are often neither obvious to the developer nor to the user of the design method. Consequently, possible potential in terms of better synthesis or optimization may often not fully be exploited.

In fact, relevant instances of any kind are often equipped with some internal (sometimes hidden) structures or properties that are unknown to the developer and/or designer [6]. One way to unveil these information is to provide a different intuition about a circuit. This can be accomplished by visualization technologies. However, existing visualization schemes for reversible circuits are basically limited to simple netlist representations in which all gates are only arranged in a cascade where black circles and $\oplus$ respectively represent control and target lines of the gates. In particular for larger circuits, these netlists do not provide a proper intuition of the structure and possible properties of reversible circuits. As an example, consider the netlist visualization of a circuit realizing a division and shown in Fig. 1 (realized by the HDL-based synthesis approach proposed in [7]). Although this circuit is composed of less than 100 gates, it is almost impossible to recognize certain structures and/or properties from this netlist visualization.

As a consequence, advanced visualization techniques are required that go beyond the straight-forward representation of a circuit as a netlist. They should mask irrelevant details as deemed necessary and, in turn, explicitly focus on highlighting the desired structures and properties. In other domains, such visualization techniques have already successfully been applied. For example:

- In the conventional software design, visualizations such as the *CodeCity* [8] are well known. Here, different software classes are placed as "buildings" within an artificial representation of a city. Depending on their properties, e.g. their number of attributes, methods, or lines of code, the ground size or the height of the "buildings" differ. Structural interrelation between classes is e.g. emphasized by placing the corresponding "buildings" in the same "district". Fig. 2a shows such a visualization taken from [8]. Unproportional looking "buildings" immediately pinpoint the designer to problematic classes in the software project. The visualization reveals classes that are too complex in terms of code and may better be split into subclasses or are not well-balanced in terms of their number of attributes to number of methods ratio.



(a) SW design (CodeCity) (b) HW debuging (c) SAT solving

Fig. 2. Visualization technologies in other domains

- In the domain of debugging (conventional) hardware, so called *error candidates* are explicitly highlighted in the netlist [9]. They represent logic elements within the circuit that may explain an erroneous behavior. Fig. 2b shows such a visualization (taken from [9]). By this, the designer is explicitly pinpointed to possible reasons for the incorrect behavior and does not have to consider all gates of the circuit at once. Furthermore, by lapping several of such layers, the designer is provided with an intuitive representation of the circuit as well as possible explanations for the error which aids him/her during the debugging process.
- Solvers for Boolean satisfiability (so called SAT solvers [10]) have been shown to be very powerful and, hence, find practical applications e.g. in domains like verification. However, although these approaches are able to efficiently solve instances composed of hundreds of thousands of variables and constraints, much smaller instances remain unsolvable within generous time limits. Understanding what makes a SAT instance hard or not has also been investigated using visualization technologies [11]. For this purpose, instances have been represented by graphs as shown in Fig. 2c (taken from [11]), where nodes represent the variables of the instance and edges the constraints over them. Using a visualization like this intuitively unveils connected sub-functions, important and less important literals, etc. This provides a better understanding about how instances could be solved in a more efficient fashion.

Motivated by these success stories, the application of visualization technologies in the domain of reversible circuit design is investigated in this work. For this purpose, we present the tool *RevVis*, a graphical interface that intuitively visualizes the structure and properties of reversible circuits. For a selected set of metrics and objectives which are relevant in the design of reversible circuits, corresponding data is collected and, afterwards, visualized in a simple fashion. The application of *RevVis* has been evaluated in a thorough case study involving several synthesis approaches that have been proposed in the past. From the different visualizations some already known structures and properties could be confirmed. Beyond that also new characteristics could be unveiled. They may be exploited in the future to further finetune these approaches and to develop corresponding new optimization schemes for the resulting circuits.

The remainder of this paper is structured as follows. The next section briefly reviews reversible circuits and some of the metrics that are considered in the following. Section 3 introduces *RevVis* and, in particular, the visualizations of the selected metrics and objectives. Afterwards, these visualizations are applied for circuits generated by several synthesis approaches. Possible conclusions drawn from that are discussed in Section 4. The paper is eventually concluded in Section 5.

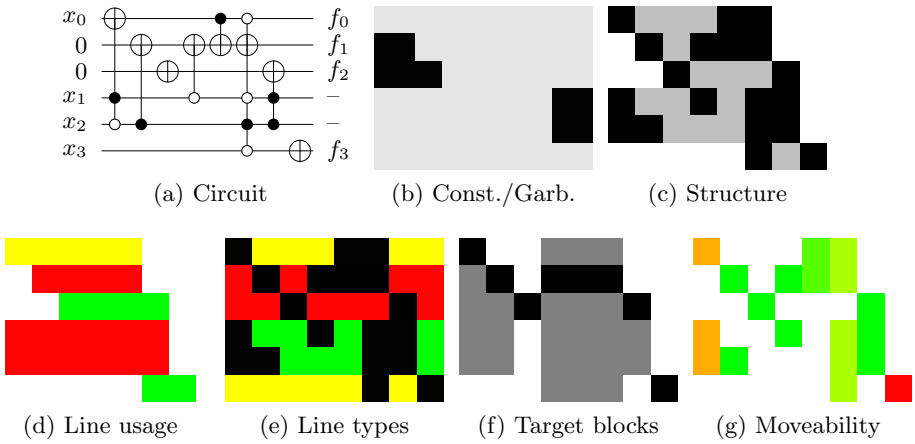
2 Background

This section briefly reviews reversible circuits as well as some of their properties which will be considered later in this paper. In general, reversible logic deals with Boolean functions which are *reversible*. A function $f : \mathbb{B}^n \to \mathbb{B}^m$ over the variables $X := \{x_1, \ldots, x_n\}$ is said to be reversible if (1) its number of inputs and outputs is equal (i.e. $n = m$) and (2) it represents a bijective, i.e. one-to-one, mapping. A reversible circuit G is composed of a cascade of reversible gates $G = g_1 g_2 \ldots g_k$ where g_i represents a reversible gate. In the past, various reversible gates such as the Toffoli gate [12], Fredkin gate [13], or Peres gate [14] have been investigated. In the context of this work, we focus on *Multiple Control Toffoli* gates which are known to be universal.

A Toffoli gate is composed of a (possibly empty) set of *control lines* $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ as well as a single *target line* $x_j \in X \setminus C$ and maps $(x_1, \ldots, x_n)$ to $(x_1, \ldots, x_{j-1}, x_j \oplus x_{i_1} \ldots x_{i_k}, x_{j+1}, \ldots, x_n)$. In other words, the logic value on the target line gets inverted if all the control inputs are at logic 1; otherwise the value on the target is passed as it is. In addition to the (positive) control lines as defined above, Toffoli gates may also be composed of negative control lines. The functionality of such gates is the same as defined above, except that the value on the target line is inverted if all values on positive control lines are assigned 1 and all values on negative control lines are assigned 0. Fig. 3a exemplarily shows a reversible circuit composed of eight Toffoli gates.

In a reversible circuit, sometimes an input line is fed with a constant logic value (0 or 1). Such circuit lines are denoted to have *constant inputs*. Similarly, circuit lines with so called *garbage outputs* may exist, i.e. circuit lines whose output value is a don't care. Garbage outputs may e.g. be needed in order to make an irreversible function reversible (see e.g. [15, 16]). The circuit from Fig. 3a has two constant inputs and two garbage outputs.

Finally, the *moving rule* for reversible circuits is partially considered in this work: Two adjacent gates g_1 and g_2 with control lines C_1 and C_2 as well as target lines t_1 and t_2, respectively, can be interchanged if $C_1 \cap \{t_2\} = \emptyset$ and $C_2 \cap \{t_1\} = \emptyset$, i.e. if none of the target lines of one gate is a control line of the other gate. Moving gates through the circuit enables further optimizations, e.g. it allows to remove or merge redundant gates (see e.g. [17–20]). Hence, the *movability* of a gate is an important metric. Consider a reversible gate sequence $G = g_1 g_2 \ldots g_k$. For every gate $g_i (1 \leq i \leq k)$, the movability of the gate is the number of possible gate positions j ($j \neq i$) such that g_i can be moved to position j according to the definition from above.



(a) Circuit (b) Const./Garb. (c) Structure

(d) Line usage (e) Line types (f) Target blocks (g) Moveability

Fig. 3. Different visualizations

3 The RevVis Tool

This section introduces the main features of the proposed visualization schemes which have been implemented in the tool *RevVis*[1]. For a selected set of metrics and objectives, the tool first collects information on the structure and properties of a given reversible circuit, which are then visualized. The visualizations are kept as simple and abstract as possible so that, even for larger designs, an intuitive and easy understanding is possible. In the following, the considered metrics and objectives are introduced. Here, all visualization schemes are illustrated by means of the reversible circuit depicted in Fig. 3a.

Constant Inputs and Garbage Outputs. Constant inputs and garbage outputs are not only essential in order to embed irreversible functions into reversible ones (see e.g. [15, 16]), but are also heavily applied in synthesis approaches e.g. based on ESOPs (e.g. [21]) or decision diagrams (e.g. [22]). Optimization approaches such as introduced in [23] rely on the fact how long circuit lines with constants or garbage are unused or not needed anymore, respectively. This is emphasized by the first visualization scheme shown in Fig. 3b. All lines inheriting a constant or garbage line are highlighted by black rows. The width of the rows depends on the number of gates in the cascade in which the respective constant (garbage) is unused (not needed anymore).

Structure of the Circuit. Reversible circuits are composed as a cascade of reversible gates which, in turn, are composed of control lines and target lines. Due to this cascade structure, the structural usage of each line in a circuit may significantly differ. This is visualized in the scheme shown in Fig. 3c. Each control

[1] RevVis is available at http://www.informatik.uni-bremen.de/agra/eng/revvis.php.

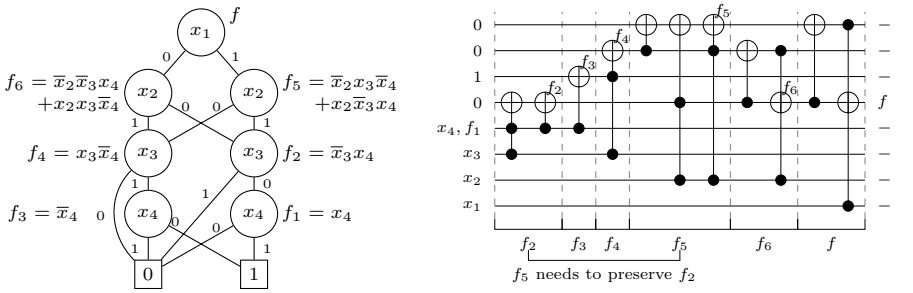
and target line connection is highlighted in black. Grey denotes the usage of each circuit line, i.e. the cascade from the first gate in which this circuit line is involved until the last gate of the cascade. White represents parts of the circuit which are not needed for the actual computation. For example, the bottom line of the considered circuit is only needed at the end of the cascade while all remaining lines are needed almost throughout the whole cascade. Although similar to the netlist visualization, this simplified view enables a more intuitive view on the structure of a circuit and can pinpoint to "holes" in the circuit (which can be used e.g. as ancilliae).

Line Usage. The usage of circuit lines is additionally visualized by the scheme shown in Fig. 3d. Here, the visualization is enriched by a color code representing the numerical usage of a circuit line. Lines highlighted red (green) represent the circuit lines with the largest (smallest) number of control and target line connections. Yellow patterns denote the circuit lines which lie between these extremes. White represents parts of the circuit which are not needed for the actual computation. Information like that could e.g. be applied for nearest neighbor optimization (see e.g. [24–27]). Here, control and target line connections always have to be adjacent, i.e. lines which are heavily used should preferably be put next to each other.

Line Types. The distribution of control and target line connections is an objective of the scheme shown in Fig. 3e. Here, red lines (green lines) denote circuit lines which are entirely composed of target lines (control lines) only; yellow lines denote circuit lines which have both control and target line connections. All actual connections are again highlighted in black. This could provide some inspiration for optimization as e.g. huge parts of the circuit composed entirely of control lines may provide some potential for reduction by factorization (see e.g. [28]).

Target blocks. Fig. 3f shows another scheme which focuses on the target line connections. More precisely, sub-circuits in which all gates have the same target line are highlighted by means of grey blocks (with the target lines additionally highlighted in black). Also this view could provide some inspiration for optimization (in particular, if the possibly different control connections could be merged so that such a cascade can be reduced to some few or even a single gate(s)).

Movability. Finally, the "movability" of gates is visualized in Fig. 3g, i.e. the applicability of the moving rule as reviewed in Section 2 is represented for each gate. Gates highlighted red have a low movability (i.e. can hardly be moved through the cascade), while gates highlighted green can be moved rather flexibly through the cascade. Obviously this view is particularly helpful to investigate optimization approaches relying on the moving rule.



(a) BDD (b) Resulting circuit

Fig. 4. BDD-based synthesis

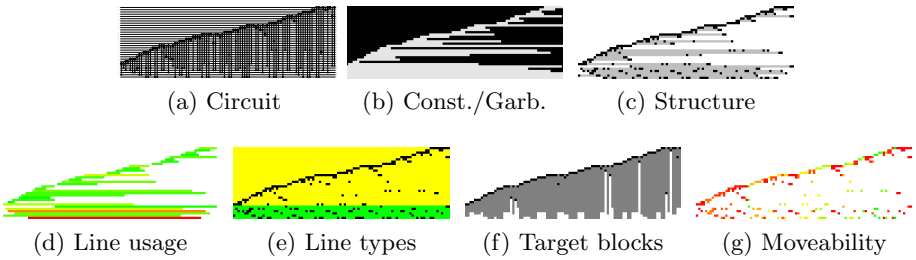
4 Applying RevVis

The visualizations proposed in the last section are supposed to provide a representation which allows to grasp a good intuition of the structure and the properties of a given circuit. In order to illustrate that *RevVis* satisfies this purpose, an intense case study has been conducted, in which circuits generated with different synthesis approaches (namely BDD-based synthesis [22], ESOP-based synthesis [21], and HDL-based synthesis [7, 29]) have been investigated using *RevVis*. In this section, results of these investigations are exemplarily shown and discussed. For this purpose, first the respective synthesis approach is briefly reviewed. Afterwards, a representative circuit (taken from RevLib [30]) is visualized and corresponding observations are discussed.

4.1 Considering Circuits Obtained by BDD-based Synthesis

The Synthesis Approach BDD-based synthesis as introduced in [22] makes use of *Binary Decision Diagrams* (BDDs) [31]. A BDD is a directed graph $G = (V, E)$ where each terminal node represents the constant 0 or 1 and each non-terminal node represents a (sub-)function. Each non-terminal node $v \in V$ has two succeeding nodes low(v) and high(v). If v is representing the function f and labeled with the variable x_i, then the corresponding sub-functions represented by the succeeding nodes are the co-factors $f_{x_i=0}$ (low(v)) and $f_{x_i=1}$ (high(v)). Thus, a BDD naturally exposes the Shannon decomposition. Having a BDD representing a function f as well as its sub-functions derived by Shannon decomposition, a reversible circuit for f can be obtained as shown by the following example.

Example 1. Fig. 4a shows a BDD representing the function $f = \overline{x}_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1\overline{x}_2x_3\overline{x}_4 + x_1x_2\overline{x}_3x_4$ as well as the respective co-factors resulting from the application of the Shannon decomposition. The co-factor f_1 can easily be represented by the primary input x_4. Having the value of f_1 available, the



(a) Circuit (b) Const./Garb. (c) Structure

(d) Line usage (e) Line types (f) Target blocks (g) Moveability

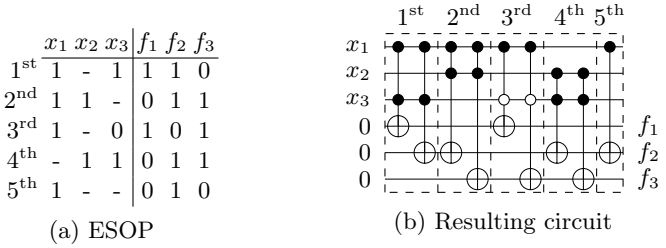
Fig. 5. Considering a circuit obtained by BDD-based synthesis

co-factor f_2 can be realized by the first two gates depicted in Fig. 4b[2]. By this, respective sub-circuits can be added for all remaining co-factors until a circuit representing the overall function f results. The remaining steps are shown in Fig. 4b.

Observations Using RevVis Fig. 5 shows the visualizations for the circuit *mod5adder_66* which has been obtained using BDD-based synthesis and works as a proper representative for this synthesis scheme. Compared to the simple netlist (see Fig. 5a), these visualizations unveil the clear structure of these circuits. In fact, BDD-based synthesis heavily relies on constant inputs (see Fig. 5b) and subsequently builds up the sub-functions (i.e. the co-factors) of the BDD. This can clearly be seen in Figs. 5c and 5f: New functionality is costantly build up towards the top-right of the circuit. The primary inputs (located at the bottom of the circuit) are frequently used for this purpose. This explains the intense usage of these circuit lines (see Fig. 5d). It also shows very nicely that the usage of the primary inputs depends on the BDD-level, e.g. the primary input represented by the root node of the BDD has a very low usage while primary inputs represented in lower levels of the BDD are accessed more often. As shown in Fig. 5e, all primary input lines are accessed in a read-only fashion (i.e. just control connections are applied in those circuit lines). Finally, Fig. 5g unveils that moveability is usually rather bad in circuits generated by BDD-based synthesis.

By this, several properties of BDD-based circuits which are already known (e.g. the huge number of constant/garbage) are confirmed. Besides that, a clearer intuition of the actual structure and properties is provided. For example, Fig. 5b may offer more precise hints where to merge constants and garbage (similar to the approach presented in [23]). Fig. 5g clearly shows that e.g. optimization approaches like template matching [17] (relying on the moving rule) are not really suitable for BDD-based circuits. Besides that, the clear stepped structure of the overall circuit might be exploitable for further optimizations.

[2] Note that an additional circuit line is added to preserve the values of x_4 and x_3 which are still needed by the co-factors f_3 and f_4, respectively.



(a) ESOP (b) Resulting circuit

Fig. 6. ESOP-based synthesis

4.2 Considering Circuits Obtained by ESOP-based Synthesis

The Synthesis Approach ESOP-based synthesis as introduced in [21] generates a reversible circuit from a Boolean function provided as *Exclusive Sum of Products* (ESOPs). ESOPs are two-level descriptions of Boolean functions that are represented as the exclusive disjunction (EXOR) of conjunctions of literals (called *products*). A *literal* is either a Boolean variable or its negation. That is, an ESOP is the most general form of two-level AND-EXOR expressions.

Having an ESOP representing a function $f : \mathbb{B}^n \to \mathbb{B}^m$, the ESOP-based synthesis approach generates a circuit with $n + m$ lines, where the first n lines work as primary inputs, while the last m circuit lines are initialized to constant 0 and work as primary outputs. Having that, Toffoli gates are selected such that the desired function is realized. This selection exploits the fact that a single product $x_{i_1} \dots x_{i_k}$ of an ESOP description directly corresponds to a Toffoli gate with control lines $C = \{x_{i_1}, \dots, x_{i_k}\}$. In case of negative literals, NOT gates or negative control lines are applied accordingly. Based on these ideas, a circuit realizing a function given as ESOP can be derived as illustrated in the following example.

Example 2. Consider the function f to be synthesized as depicted in Fig. 6a[3]. The first product x_1x_3 affects f_1 and f_2. Hence, two Toffoli gates which have target lines f_1 and f_2 and control lines $C = \{x_1, x_3\}$ are added (see Fig. 6b). The third product $x_1\overline{x}_3$ includes a negative literal. Thus, the Toffoli gates added for this product have a negative control line on x_3. This procedure is continued until all products have been considered. The resulting circuit is shown in Fig. 6b.

Observations Using RevVis Fig. 7 shows the visualizations for the circuit *rd73_252* which has been obtained using ESOP-based synthesis and works as a proper representative for this synthesis scheme. Compared to the simple

[3] The column on the left-hand side gives the products, where a "1" on the i^{th} position denotes a positive literal (i.e. x_i) and a "0" denotes a negative literal (i.e. $\overline{x}_i$), respectively. A "–" denotes that the respective variable is not included in the product. The right-hand side gives the primary output patterns.

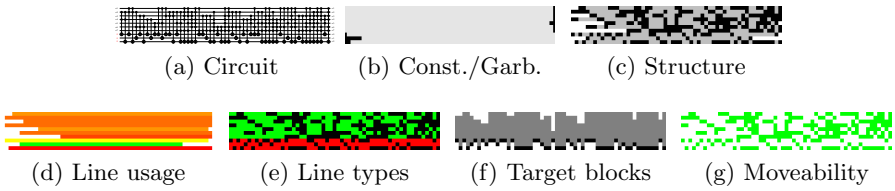


Fig. 7. Considering a circuit obtained by ESOP-based synthesis

netlist (see Fig. 7a), the characteristic structure is clearly unveiled thanks to the visualizations. In particular, the distinction between input lines (which have control connections only) and output lines (which have target connections only) becomes evident (see Fig. 7e) and also leads to a very regular structure with respect to target blocks (see e.g. Fig. 7f). This provides potential as it may allow to merge gates with equal control lines but different target lines (as discussed e.g. in [19]). Furthermore, approaches relying on the moving rule (e.g. [17]) significantly benefit from this structure as it leads to a very high movability (see Fig. 7g). It may also be observed that, due to the high movability of gates, many target blocks can be merged leading to more potential for optimization. In contrast, constant inputs are used very early in the cascade (see Fig. 7b), i.e. there is no potential to reduce the number of constant/garbage lines using e.g. the method proposed in [23]. Besides that, ESOP-based circuits seem to have a rather irregular structure, i.e. the respective gate connections are distributed rather arbitrarily (see Fig. 7c). However, it can be observed that inputs lines are used more often than output lines (see Fig. 7d). This can be explained by the fact that some factors may have to be applied to several functions and, hence, identical control connections are frequently applied.

4.3 Considering Circuits Obtained by HDL-based Synthesis

The Synthesis Approach The strive for more scalable synthesis approaches also led to the definition and consideration of a *Hardware Description Language* (HDL) for reversible circuits in [7]. In order to ensure reversibility in the description, this HDL distinguishes between reversible assignments (denoted by $\oplus=$) and not necessarily reversible *binary operations* (denoted by $\odot$). The former class of operations assigns values to a signal on the left-hand side. Therefore, the left-hand side signal must not appear in the expression on the right-hand side. Furthermore, only a restricted set of assignment operations exists, namely increase (`+=`), decrease (`-=`), and bit-wise XOR (`^=`). These operations preserve the reversibility (i.e. it is possible to compute these operations in both directions). In contrast, binary operations, e.g. arithmetic, bit-wise, logical, or relational operations, may not be reversible and, hence, can only be used in right-hand expressions which preserve the values of the inputs. In doing so, all computations remain reversible since the input values can be applied to reverse any operation. For example, to describe a multiplication (i.e. `a*b`), a new free

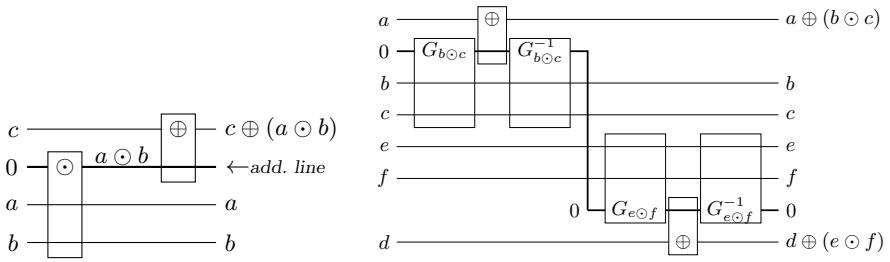


Fig. 8. HDL-based synthesis

signal `c` must be introduced which is used to store the product (i.e. `c^=a*b` is applied). In comparison to common (non-reversible) languages, this forbids statements like `a=a*b`.

Having such an HDL description, synthesis approaches like introduced in [7] generate corresponding circuits following a hierarchical scheme. That is, existing realizations of the individual operations (i.e. building blocks) are combined so that the desired circuit is realized. This is illustrated in Fig. 8a for the generic operation $c\oplus = (a\odot b)$. First, the binary operation $\odot$ is realized (using additional circuit lines with constant inputs). Afterwards, the intermediate result is utilized to realize the complete statement including its reversible assignment $\oplus=$.

This scheme has further been improved in [29]. Here, the values of intermediate results are reversed once they are not needed any longer (leading back to the original constant value). Then, no new additional lines might be required to buffer upcoming intermediate results. The general idea is briefly illustrated in Fig. 8b by means of the generic HDL statements $a\oplus = (b\odot c)$ and $d\oplus = (e\odot f)$. First, two sub-circuits $G_{b\odot c}$ and $G_{a\oplus=b\odot c}$ are added ensuring that the first statement is realized. This is equal to the procedure from Fig. 8a and leads to additional lines with constant inputs. But then, a further sub-circuit $G^{-1}_{b\odot c}$ is applied. Since $G^{-1}_{b\odot c}$ is the inverse of $G_{b\odot c}$, this sets the circuit lines buffering the result of $b\odot c$ back to the constant 0. As a result, these circuit lines can be reused in order to realize the following statements as illustrated for $d\oplus=e\odot f$ in Fig. 8b.

Observations Using RevVis Fig. 9 (Fig. 10) shows the visualizations for the circuit *mult_stmts_3bit* which has been obtained using the straight-forward HDL-based synthesis as illustrated in Fig. 8a (the improved HDL-based synthesis as illustrated in Fig. 8b) and works as a proper representative for this synthesis scheme. More precisely, these circuits realize three HDL-statements over 3-bit variables. The respective cascades for each statement are separated by vertical lines in Fig. 9 and Fig. 10. Compared to the simple netlist (see Fig. 9a and Fig. 10a), these visualizations do not only unveil the structure and characteristics of the respective circuits, but also the differences between the straight-forward and optimized synthesis scheme.

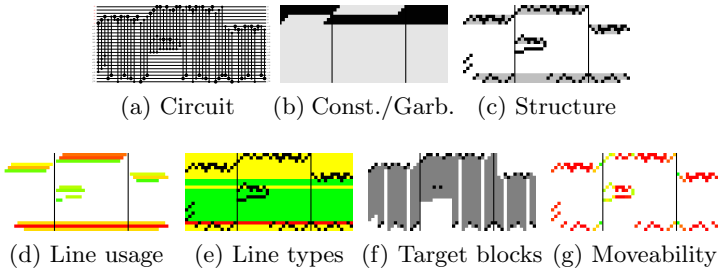


Fig. 9. Considering a circuit obtained by HDL-based synthesis

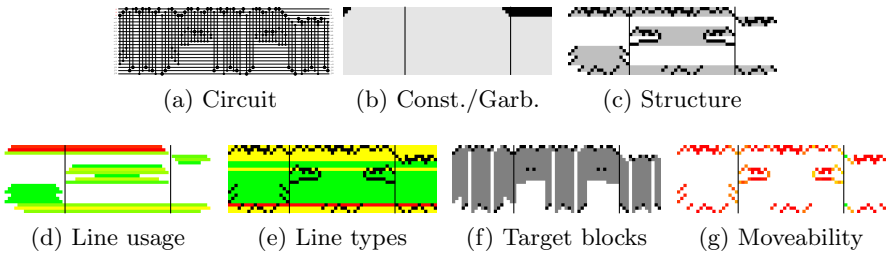


Fig. 10. Considering a circuit obtained by improved HDL-based synthesis

First of all, the structures sketched in Fig. 8, i.e. the building blocks for binary operations, reversible assignments, and reversing, can also be recognized in the visualizations (see e.g. Fig. 9c and Fig. 10c). In particular for the improved scheme, the symmetry resulting from reversing intermediate results is rather obvious. Here, it can also be observed that just one set of constant circuit lines is needed, while the straight-forward approach uses several constant circuit lines only for a short time (compare Fig. 9b and Fig. 10b). The frequent re-use of these lines in the improved approach is also reflected in the line usage visualization (see Fig. 10d).

Besides that, many circuit lines only have control connections in this example (see Figs. 9e and 10e). This is caused by the fact that the three HDL-statements are of the form $a \oplus = (b \odot c)$, i.e. b and c never occur on the left-hand side of a statement. Finally, the visualization clearly unveils that HDL-based circuits have a rather poor moveability and, hence, do not seem very suitable for optimization schemes such as [17] (see Figs. 9g and 10g).

5 Conclusions

In this work, we considered the visualization of reversible circuits. This is motivated by the fact that certain structures and properties of circuits are often not obvious to the developer or to the user. Furthermore, simple netlist representations do not provide a proper intuition and, hence, are not suitable – particularly for circuits of larger size. In order to address this, we introduced the tool *RevVis* which provides visualization layers for several metrics as well as objectives and, by this, intuitively highlights structures and properties of reversible circuits. The application of *RevVis* has been evaluated in a thorough case study involving several synthesis approaches. This enabled a deeper discussion about both known as well as new characteristics of the obtained circuits and, hence, the considered synthesis schemes. In the future, visualizations as proposed in this work will be beneficial to draw conclusions from newly developed design approaches right from the beginning as well as to gain inspiration for new synthesis and optimization methods.

References

1. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
2. Berut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., Lutz, E.: Experimental verification of Landauer's principle linking information and thermodynamics. Nature **483** (2012) 187–189
3. Wille, R., Drechsler, R., Osewold, C., Garcia-Ortiz, A.: Automatic design of low-power encoders using reversible circuit synthesis. In: Design, Automation and Test in Europe. (2012) 1036–1041
4. Drechsler, R., Wille, R.: From truth tables to programming languages: progress in the design of reversible circuits. In: Int'l Symp. on Multi-Valued Logic. (2011) 78–85
5. Saeedi, M., Markov, I.L.: Synthesis and optimization of reversible circuits - a survey. ACM Computing Surveys **45**(2) (2011)
6. Walsh, T.: Search in a small world. In: International Conference on AI. (1999) 1172–1177
7. Wille, R., Offermann, S., Drechsler, R.: SyReC: A programming language for synthesis of reversible circuits. In: Forum on Specification and Design Languages. (2010) 184–189
8. Wettel, R., Lanza, M., Robbes, R.: Software systems as cities: a controlled experiment. In: International Conference on Software Engineering. (2011) 551–560
9. Sülflow, A., Wille, R., Genz, C., Fey, G., Drechsler, R.: FormED: A formal environment for debugging. In: University Booth at the Design, Automation and Test in Europe. (2009)
10. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Conference on Theory and Applications of Satisfiability Testing, Springer (2003) 502–518
11. Sinz, C.: Visualizing SAT instances and runs of the DPLL algorithm. J. Autom. Reasoning **39**(2) (2007) 219–243
12. Toffoli, T.: Reversible computing. In de Bakker, W., van Leeuwen, J., eds.: Automata, Languages and Programming. Springer (1980) 632 Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

13. Fredkin, E., Toffoli, T.: Conservative logic. Int'l Journal of Theoretical Physics **21**(3–4) (1982) 219–253
14. Peres, A.: Reversible logic and quantum computers. Phys. Rev. A **32**(6) (1985) 3266–3276
15. Maslov, D., Dueck, G.W.: Reversible cascades with minimal garbage. Trans. on CAD **23**(11) (2004) 1497–1509
16. Wille, R., Keszöcze, O., Drechsler, R.: Determining the minimal number of lines for large reversible circuits. In: Design, Automation and Test in Europe. (2011) 1204–1207
17. Miller, D.M., Maslov, D., Dueck, G.W.: A transformation based algorithm for reversible logic synthesis. In: Design Automation Conf. (2003) 318–323
18. Maslov, D., Dueck, G.: Quantum circuit simplification and level compaction. Trans. on CAD **27**(3) (2008) 436–444
19. Wille, R., Soeken, M., Otterstedt, C., Drechsler, R.: Improving the mapping of reversible circuits to quantum circuits using multiple target lines. In: ASP Design Automation Conf. (2013)
20. Datta, K., Rathi, G., Wille, R., Sengupta, I., Rahaman, H., Drechsler, R.: Exploiting negative control lines in the optimization of reversible circuits. In: International Conference on Reversible Computation. (2013) 209–220
21. Fazel, K., Thornton, M.A., Rice, J.E.: ESOP-based Toffoli gate cascade generation. In: Pacific Rim Conference on Communications, Computers and Signal Processing. (2007) 206–209
22. Wille, R., Drechsler, R.: BDD-based synthesis of reversible logic for large functions. In: Design Automation Conf. (2009) 270–275
23. Wille, R., Soeken, M., Drechsler, R.: Reducing the number of lines in reversible circuits. In: Design Automation Conf. (2010) 647–652
24. Saeedi, M., Wille, R., Drechsler, R.: Synthesis of quantum circuits for linear nearest neighbor architectures. Quantum Information Processing **10**(3) (2011) 355–377
25. Alfailakawi, M., Alterkawi, L., Ahmad, I., Hamdan, S.: Line ordering of reversible circuits for linear nearest neighbor realization. Quantum Information Processing **12**(10) (2013) 3319–3339
26. Shafaei, A., Saeedi, M., Pedram, M.: Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In: Design Automation Conf. (2013) 41
27. Wille, R., Lye, A., Drechsler, R.: Optimal SWAP gate insertion for nearest neighbor quantum circuits. In: ASP Design Automation Conf. (2014) 489–494
28. Miller, D.M., Wille, R., Drechsler, R.: Reducing reversible circuit cost by adding lines. In: Int'l Symp. on Multi-Valued Logic. (2010)
29. Wille, R., Soeken, M., Schönborn, E., Drechsler, R.: Circuit line minimization in the HDL-based synthesis of reversible logic. In: Annual Symposium on VLSI. (2012) 213–218
30. 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.
31. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. Trans. on Comp. **35**(8) (1986) 677–691