From Truth Tables to Programming Languages: Progress in the Design of Reversible Circuits

Rolf Drechsler Robert Wille

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany

{drechsle,rwille}@informatik.uni-bremen.de

Abstract—**It is a widely supported prediction that conventional computer hardware technologies are going to reach their limits in the near future. Consequently, researchers are working on alternatives. Reversible circuits are one promising direction with applications e.g. in low-power design or quantum computation. However, no real design flow for this new kind of circuits exists so far.**

In this paper, the progress in the development of design methods for reversible circuits is reviewed – with a particular focus on the synthesis steps. After a brief review on reversible circuits, the general idea of common synthesis approaches is described. This includes methods based on truth table descriptions, methods applicable to larger functions, and finally an approach based on a programming language. Discussions and an outlook to future work conclude this paper.

I. Introduction

In the recent years, reversible computation established itself as a very promising research area and an emerging technology. This is motivated by a widely supported prediction that the conventional computer hardware technologies are going to reach their limits in the near future.

In particular, shrinking transistor sizes and power dissipation are the major barriers in the development of smaller and more powerful circuits. Here, a fundamental limitation of conventional computing becomes evident: Each time information is lost, energy is dissipated regardless of the underlying technology (also known as Landauer's principle which has already been observed in 1961 [1]). While the amount of energy dissipation caused by this was negligible in the last decades (in fact, only $k \cdot T \cdot \log 2$ Joules of energy are dissipated for each "lost" bit of information, where k is the Boltzmann constant and T is the temperature), it may become crucial considering that (1) in today's circuitry millions of operations are performed in a single second and (2) more operations are performed with smaller transistor sizes (i.e. on a smaller area).

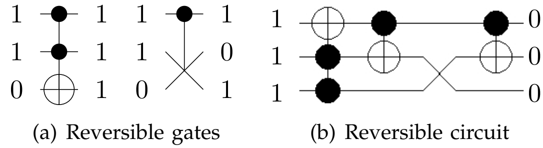
As a consequence, Landauer (and later Bennett [2], Fredkin [3], Toffoli [4], and others) suggested the use of reversible circuits, i.e. circuits with an equal number of input and output signals, whereby each input assignment maps to a unique output assignment (i.e. the function represented by the circuit is a bijection). Since reversible circuits are by definition information-lossless, power dissipation resulting from Landauer's principle, as described above, can be decreased or even eliminated – the fundamental limit is evaded.

Besides that, quantum computation [5] has become a major application area for reversible logic. It uses qubits instead of the conventional bits. Qubits allow to represent not only 0 and 1 but also a superposition of both. As a result, qubits can represent multiple states at the same time enabling theoretically enormous speedups in computation. It has been shown that, for example, using a quantum circuit it is possible to solve the factorization problem in polynomial time while for conventional circuits only exponential methods exist [6].

For both application scenarios, first physical implementations exist confirming the promising assumptions and motivating further research (see e.g. [7] for low-power design or [8] for quantum circuits). Besides that, reversible logic additionally finds application in domains like optical computing [9], DNA computing [10], as well as nanotechnologies [11]. Also, cryptography or encoding/decoding methods (e.g. for music and videos) can profit from enhancements in this area (see e.g. [12]). Furthermore, already today reversible operations are used in instruction sets for microprocessors [13].

However, no design flow for this new kind of circuits exists so far. This is crucial since the design of reversible and quantum systems significantly differs from their conventional counterparts. Many concepts and methods developed for conventional hardware design in the last decades have to be redeveloped in order to support the new technologies. Accordingly, researchers started working on such a design flow already some years ago.

In this paper, the progress of this work is reviewed – with a particular focus on the synthesis steps. Here, significant achievements have been made in the last years. While the first approaches were entirely based on truth table descriptions of the function to be synthesized, recently a first programming language for reversible circuit design has been introduced. We provide an overview of these different methods and discuss the respective advantages as well as the still open problems. The descriptions are thereby kept brief and focus on illustrating the general idea, respectively. For a more



(a) Reversible gates (b) Reversible circuit

Fig. 1. Reversible circuitry

detailed treatment, references to the respective original work are provided. Moreover, most of the approaches summarized in this paper are also publicly available in RevKit [14].

The remainder of the paper is thereby structured as follows: First the basics on reversible circuits are introduced. Afterwards, the common synthesis approaches are reviewed. Starting with the initial approaches based on truth table descriptions (Section III), methods applicable to larger functions (Section IV), and finally an approach based on a programming language (Section VI) are described. At the end of the paper, conclusions are drawn and an outlook to future research is provided.

II. Reversible Circuits

Reversible circuits are digital circuits with the same number of input signals and output signals. Furthermore, reversible circuits realize bijections only, i.e. each input assignment maps to a unique output assignment. Accordingly, computations can be performed in both directions (from the inputs to the outputs and vice versa).

Reversible circuits are composed as cascades of reversible gates. Each reversible gate over the inputs $X = \{x_1, \ldots, x_n\}$ consists of a (possibly empty) set $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ of *control lines* and a set $T \subset X \setminus C$ of *target lines*. Commonly used reversible gates are:

- The *Toffoli gate* $TOF(C, x_t)$ [4], which consists of a single target line $x_t \in X \setminus C$ whose value is inverted if all values on the control lines are set to 1 or if $C = \emptyset$, respectively. All remaining values are passed through the gate unaltered.
- The Fredkin gate $F(C, \{x_{t1}, x_{t2}\})$ [3] which consists of two target lines $x_{t1}, x_{t2} \in X \setminus C$ interchanges the values of these target lines if all values on the control lines are set to 1 or if $C = \emptyset$, respectively. Again, all remaining values are passed through the gate unaltered.

Example 1. *Fig. 1(a) shows a Toffoli gate with two control lines and a Fredkin gate with one control line, respectively. The control lines are denoted by ●, while the target lines are denoted by $\oplus$ (for the Toffoli gate) or $\times$ (for the Fredkin gate), respectively. The annotated values demonstrate the computation of the respective gates. Fig. 1(b) shows different reversible gates in a cascade forming a reversible circuit.*

TABLE I
Quantum cost for Toffoli and Fredkin gates

No. of control lines	Quantum cost of a Toffoli gate	Quantum cost of a Fredkin gate
0	1	3
1	1	7
2	5	15
3	13	28, if at least 2 lines are unconnected 31, otherwise
4	26, if at least 2 lines are unconnected 29, otherwise	40, if at least 3 lines are unconnected 54, if 1 or 2 lines are unconnected 63, otherwise
5	38, if at least 3 lines are unconnected 52, if 1 or 2 lines are unconnected 61, otherwise	52, if at least 4 lines are unconnected 82, if 1, 2 or 3 lines are unconnected 127, otherwise

To measure the cost of a reversible circuit, different metrics are applied (sometimes depending on the addressed technology). In general, the number of circuit lines is an important criterion. In particular in the domain of quantum computation, the number of lines is equal to the number of qubits – so far a very restricted resource.

Beyond that, the costs of the respective gates themselves are important, too. Since simply counting the number of gates does not adequately reflect the effort to realize them, so called *quantum costs* are applied. They measure how many elementary quantum operations are needed in order to realize a reversible gate [5]. In the past different methods have been introduced that convert a reversible gate into its equivalent quantum operations (see e.g. [15], [16]). Accordingly, different metrics for quantum costs are applied. In this work, we use the metric shown in Table I. As can be seen, the costs depend thereby on the number of control lines, a reversible gate consists of.

Example 2. *The circuit shown in Fig. 1(b) consists of three circuit lines and has quantum cost of 10.*

III. Truth Table-based Synthesis

One of the first synthesis approaches for reversible circuits relies on truth table descriptions of the function to be synthesized. The given functions often need thereby to be reversible. Since this is not the case for many practical functions, a pre-processing step called *embedding* often is performed first. This creates a reversible description of the given function which afterwards can be used to realize the desired circuit. In this section, both steps, i.e. the embedding as well as the actual synthesis, of functions given in terms of truth tables are exemplarily reviewed.

TABLE II
ADDER FUNCTION AND A POSSIBLE EMBEDDING

(a) Adder function

c_{in}	x	y	c_{out}	sum	
0	0	0	0	0	0
0	0	1	**0**	**1**	**0**
0	1	0	**0**	**1**	**1**
0	1	1	*1*	*0*	*0*
1	0	0	**0**	**1**	**?**
1	0	1	*1*	*0*	*1*
1	1	0	*1*	*0*	*?*
1	1	1	1	1	1

(b) Embedding

0	$\mathbf{c}_{in}$	**x**	**y**	$\mathbf{c}_{out}$	**sum**	g_1	g_2
0	**0**	**0**	**0**	**0**	**0**	0	0
0	**0**	**0**	**1**	**0**	**1**	1	1
0	**0**	**1**	**0**	**0**	**1**	1	0
0	**0**	**1**	**1**	**1**	**0**	0	1
0	**1**	**0**	**0**	**0**	**1**	0	0
0	**1**	**0**	**1**	**1**	**0**	1	1
0	**1**	**1**	**0**	**1**	**0**	1	0
0	**1**	**1**	**1**	**1**	**1**	0	1
1	0	0	0	1	0	0	0
				…			

TABLE III
TRANSFORMATION-BASED METHOD

line (i)	input abcd	output abcd	1st abcd	2nd abcd	3rd abcd	4th abcd	5th abcd	6th abcd
0	0000	0000	0000	0000	0000	0000	0000	0000
1	0001	0111	0101	0001	0001	0001	0001	0001
2	0010	0110	0110	0110	0010	0010	0010	0010
3	0011	1001	1011	1111	1011	0011	0011	0011
4	0100	0100	0100	0100	0100	0100	0100	0100
5	0101	1011	1001	1101	1101	1101	0101	0101
6	0110	1010	1010	1010	1110	1110	1110	0110
7	0111	1101	1111	1011	1111	0111	1111	0111
8	1000	1000	1000	1000	1000	1000	1000	1000
9	1001	1111	1101	1001	1001	1001	1001	1001
10	1010	1110	1110	1110	1010	1010	1010	1010
11	1011	0001	0011	0111	0011	1011	1011	1011
12	1100	1100	1100	1100	1100	1100	1100	1100
13	1101	0011	0001	0101	0101	0101	1101	1101
14	1110	0010	0010	0010	0110	0110	0110	1110
15	1111	0101	0111	0011	0111	1111	0111	1111

A. Embedding of Irreversible Functions

The embedding process is illustrated by means of the adder function shown in Table II(a). The adder has three inputs (the carry-bit c_{in} as well as the two summands x and y) and two outputs (the carry c_{out} and the *sum*). The adder obviously is not reversible (irreversible), since (1) the number of inputs differs from the number of outputs and (2) there is no unique input-output mapping. Even adding an additional output to the function (leading to the same number of input and outputs) would not make the function reversible. Then, the first four lines of the truth table can be embedded with respect to reversibility as shown in the rightmost column of Table II(a). However, since $c_{out} = 0$ and $sum = 1$ already appeared two times (marked bold), no unique embedding for the fifth truth table line is possible any longer. The same also holds for the lines marked italic.

This already has been observed in [17] and was further discussed in [18]. There, the authors came to the conclusion that at least $\lceil log(m) \rceil$ free outputs are required to make an irreversible function reversible, where m is the maximum number of times an output pattern is repeated in the truth table. Since for the adder at most 3 output pattern are repeated, $\lceil log(3) \rceil = 2$ free outputs (and therewith one additional circuit line) are required to make the function reversible.

Adding new lines causes constant inputs and garbage outputs. The value of the constant inputs can be chosen by the designer. Garbage outputs are by definition don't cares and thus can be left unspecified leading to an incompletely specified function. However, many synthesis approaches require a completely specified function so that all don't cares must be assigned with a concrete value.

As a result, the adder is embedded in a reversible function including four variables, one constant input, and two garbage outputs. A possible assignment to the constant as well as the don't care values is depicted in Table II(b). Note that the concrete embedding may influence the respective synthesis results. Corresponding evaluations have been made e.g., in [19], [20].

B. Synthesis Using the Transformation-based Approach

With a reversible function (given in terms of a truth table) at hand, the synthesis can be performed. In the following this is illustrated by means of the transformation-based approach introduced in [21]. Here, the basic idea is to traverse each line of the truth table and to add gates to the circuit until the output values match the input values (i.e. until the identity of both is achieved). Gates are thereby chosen so that they don't alter already considered lines. Furthermore, gates are added starting at the output side of the circuit (this is, because *output* values are transformed until the identity is achieved).

In the following, we describe the respective steps of the approach using the (embedded) adder function from above and Table III. The first column of Table III denotes the respective line numbers, while the second and third column give the function specification of the adder (taken from Table II(b)). For brevity, the inputs 0, c_{in}, x, y and outputs c_{out}, sum, g_1, g_2 are denoted by a, b, c, d, respectively. The remaining columns provide the transformed output values for the respective steps.

The algorithm starts at truth table line 0. Since for this line the input is equal to the output (both are assigned to 0000), no gate has to be added. In contrast, to match the output with the input in truth table line 1, the values for c and b must be inverted. To this end, two gates $TOF(\{d\}, c)$ (1st step) and $TOF(\{d\}, b)$ (2nd step) are added as depicted in Fig. 2. Because of the control line d, this does not affect the previous truth table line. In line 2 and line 3, a $TOF(\{c\}, b)$ gate as well as a $TOF(\{c, d\}, a)$ gate is needed to match the values of b and a, respectively (step 3 and 4). For the latter, two control lines are needed to keep the already traversed truth table lines unaltered. Afterwards, only two more gates $TOF(\{d, b\}, a)$ (5th step) and $TOF(\{c, b\}, a)$ (6th step) are necessary to achieve the input-output identity. The resulting circuit is shown in Fig. 2. This circuit consists of six gates and has quantum cost of 18.

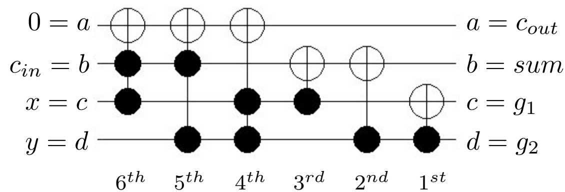


Fig. 2. Circuit obtained by transformation-based synthesis

In [21], further variations of this approach are discussed. In fact, this transformation can also be applied in the inverse direction (i.e., so that the input must match the output) and in both directions simultaneously. Furthermore, in [22] the approach has been extended by the application of templates. Templates help one to reduce the size of the resulting circuits and, thus, to achieve circuits with lower cost.

C. Further Approaches and Discussion

While the method described above represent only one possible way to synthesize reversible functions, many other approaches exist as well. One can distinguish between further heuristic approaches and exact methods. The strategy introduced in the last section (namely selecting reversible gates so that the chosen function representation becomes the identity) was thereby often adopted and extended. More compact data-structures like decision diagrams [23], positive-polarity Reed-Muller expansion [24], or Reed-Muller spectra [25] have been utilized for this purpose. Besides that, also complementary approaches have been introduced (e.g. [26], [27]). Instead, exact approaches often are based on simple enumeration or apply formal methods (see e.g. [28], [29]).

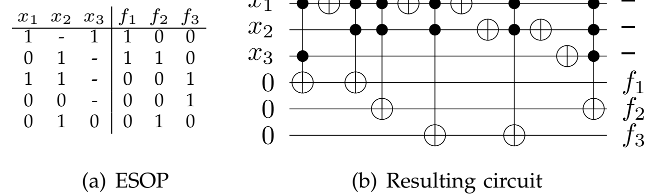
However, the scalability of all these approaches is limited, i.e. the methods are applicable for relatively small functions only. Exact approaches reach their limits with functions containing more than 6 variables [29] while heuristic methods are able to synthesize functions with at most 30 variables [24]. Moreover, often a significant amount of run-time is needed to achieve these results. This is mainly caused by the chosen function representation. If larger functions should be synthesized, more compact function descriptions and, accordingly, other synthesis approaches have to be considered.

IV. Boolean Synthesis for Large Functions

To overcome the limitations of synthesis approaches based on truth tables (or similar function representations), alternatives exploiting exclusive sum of products and decision diagrams have been introduced. These methods are reviewed in this section.

A. ESOP-based Synthesis

The first approach presented here to illustrate synthesis of large functions makes use of *Exclusive Sum of Products* (ESOPs) and has been proposed in [30]. ESOPs are two-level descriptions of Boolean functions. Each ESOP is composed of various conjunctions of literals (called *products*). A *literal* either is a propositional variable or its negation. To form the ESOP, all products are combined by Exclusive ORs. That is, an ESOP is the most general form of two-level AND-EXOR expressions.



x_1	x_2	x_3	f_1	f_2	f_3
1	-	1	1	0	0
0	1	-	1	1	0
1	1	-	0	0	1
0	0	-	0	0	1
0	1	0	0	1	0

(a) ESOP (b) Resulting circuit

Fig. 3. ESOP-based synthesis

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

Example 3. *Consider the function f to be synthesized as depicted in Fig. 3(a)*[1]*. The first product x_1x_3 affects f_1. Accordingly, a Toffoli gate with control lines $C = \{x_1x_3\}$ and a target line representing the primary output f_1 is added (see Fig. 3(b)). The next product $\overline{x}_1x_2$ includes a negative literal. Thus, a NOT gate is needed at line x_1 to generate the appropriate value for the next mappings. Since $\overline{x}_1x_2$ affects both, f_1 and f_2, two Toffoli gates with control lines $C = \{x_1x_2\}$ are added next. Afterwards, a further NOT gate is applied to restore the value of x_1 (needed again by the third product). This procedure is continued until all products have been considered. The resulting circuit is shown in Fig. 3(b).*

Note that thereby the order in which the respective products are traversed may have a significant impact on the resulting circuit cost. For example, the line x_1 in the circuit from Example 3 is unnecessarily often inverted. This can be avoided by treating the respective products in a different order. Improvements of the original approach exploiting these (and other) observations have been proposed e.g. in [31].

[1]The column on the left-hand side gives the respective products, where a "1" on the ith 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 respective primary output patterns.

B. BDD-based Synthesis

Another alternative aimed at synthesizing large functions as reversible circuit has been proposed in [32]. Here, *Binary Decision Diagrams* (BDDs) [33] are exploited. 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 thereby two succeeding nodes $\mathrm{low}(v)$ and $\mathrm{high}(v)$. 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 $f_{x_i=0}$ $(\mathrm{low}(v))$ and $f_{x_i=1}$ $(\mathrm{high}(v))$. 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.

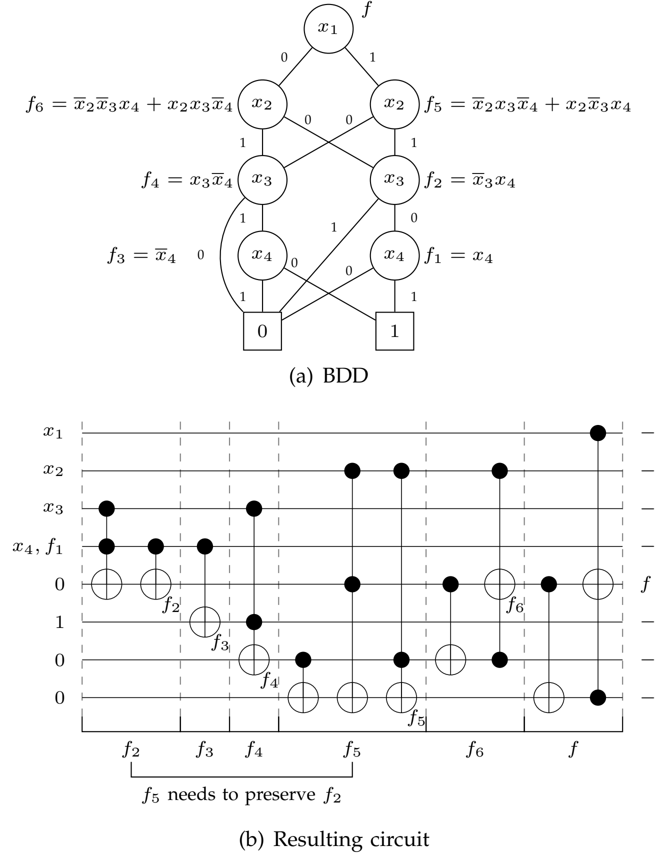
Example 4. *Fig. 4(a) shows a BDD representing the function* $f = \overline{x}_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1\overline{x}_2x_3\overline{x}_4 + x_1x_2\overline{x}_3x_4$ *as well as the respective co-factors resulting from the application of the Shannon decomposition. The co-factor* f_1 *can easily be represented by the primary input* x_4*. Having the value of* f_1 *available, the co-factor* f_2 *can be realized by the first two gates depicted in Fig. 4(b)*[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. 4(b).*

That is, to realize (possibly large) functions, decomposition is applied leading to smaller sub-functions for which existing building blocks can be applied. Then, the resulting sub-circuits can be composed to realize the overall function.

As can be seen, this method sometimes requires additional circuit lines with constant inputs in order to preserve (temporary) values. For example, as already shown above, an additional line is required to realize the co-factor f_2 without losing the input value x_4 (which is still needed to realize f_3). A similar issue occurs for the co-factor f_5. Here, the values of f_2 and f_4 have to be preserved since they are still needed later to realize co-factor f_6.

Further improvements can be achieved if, advanced BDD techniques like re-ordering or complement edges are exploited. Also the use of further decomposition types result in improvements. In particular, the positive Davio decomposition is thereby promising, since the respective transformation can directly be realized by a single Toffoli gate. Improvements based on these ideas have been reported in [34], [35].

[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) BDD

(b) Resulting circuit

Fig. 4. Example for BDD-based synthesis

V. Synthesis Using Programming Languages

The approaches reviewed in the last section are capable to realize circuits for functions with more than 100 variables. However, they still rely on Boolean descriptions and, thus, do not allow the design of complex reversible systems. Consequently, higher levels of abstractions have been considered leading to the development of hardware description languages. A first version of such a language called *SyReC* has recently been introduced in [36].

A. The SyReC Language

SyReC is based on the reversible software language *Janus* [37], which has been enriched by further concepts (e.g. declaring circuit signals of different bit-widths), new operations (e.g. bit-access and shifts), and some restrictions (e.g. the prohibition of dynamic loops). Fig. 5 shows a typical SyReC program specifying a simple arithmetic logic unit.

As can be seen, a SyReC program includes the declaration of modules and signals of the circuit to be specified (Line 1). Signals represent thereby non-negative integers as its sole data type. Besides that, different statements and expressions are available to specify the functionality of the circuit. In order to ensure the reversibility, these

```
1  module alu( in op(2), in a, in b, out c )
2    if ( op = 0 ) then
3      c ^= ( a + b )
4    else
5      if ( op = 1 ) then
6        c ^= ( a − b )
7      else
8        if ( op = 2 ) then
9          c ^= ( a ∗ b )
10       else
11         c ^= ( a / b )
12       fi ( op = 2 )
13     fi ( op = 1 )
14   fi ( op = 0 )
```

Fig. 5. SyReC specification of an ALU

statements must satisfy certain criteria. For example, each conditional statement (i.e. each if-statement) has to be terminated by a corresponding fi-statement (see e.g. Line 12). Furthermore, statements and expressions are distinguished between *reversible assignment operations* and not necessarily reversible *binary operations*.

Reversible assignment operations assign values to a signal on the left-hand side. Therefore, the respective signal must not appear in the expression on the right-hand side. Furthermore, only a restricted set of assignment operations exists, namely increase (+=), decrease (-=), bitwise 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. Thus, they can only be used in right-hand expressions which preserve, i.e. do not modify, the values of the respective inputs. In doing so, all computations remain reversible since the input values can be applied to reverse any operation. For example, to specify the multiplication in Line 9, a new free signal c must be introduced which is used to store the product. In comparison to common (irreversible) programming languages, statements such as a=$b + (5 * a)$ are not allowed.

For a detailed introduction of the SyReC programming language, we refer to [36] as well as to the detailed documentation provided at the RevLib benchmark webpage [38].

B. Synthesis of SyReC Specifications

Using the SyReC language, it is possible to design reversible circuits on a higher level. For example, the arithmetic logic unit can be specified much easier using the code from Fig. 5 in comparison to methods based on truth tables or decision diagrams. However, the specified circuits still need to be synthesized.

Therefore, a hierarchical synthesis method has been introduced in [36]. This approach uses existing realizations of the individual operations (i.e. building blocks) and combines them so that the desired circuit results. Previous work on the realization of these building blocks (e.g. [39], [40]) was thereby exploited.

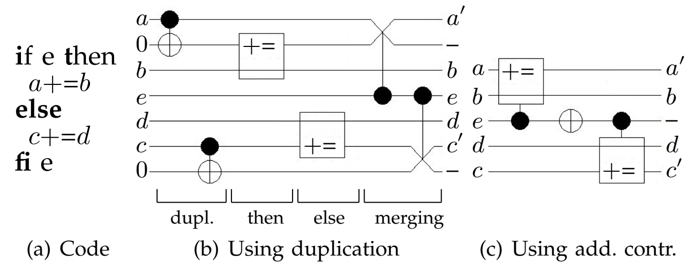


Fig. 6. Realization of an if-statement

Besides that, the realization of control logic was not investigated in detail before and, thus, received special attention. The realization of loops and module calls is thereby straightforward, since the respective instructions simply have to be cascaded together. In contrast, *conditional statements* require more elaborated methods. In [36], the two approaches depicted in Fig. 6 have been introduced for this purpose:

1) The first one (shown in Fig. 6(b)) relies on duplication. Here, the values of all signals that possibly might be affected in an if- or else-block are copied (using an additional circuit line with a constant input as shown by Signal a and Signal c in Fig. 6). Then, sub-circuits realizing the respective if-/else-block are added (denoted by the boxes). Finally, depending on the result of the conditional statement (Signal e in Fig. 6), the values of the duplicated lines and the original lines are swapped leading to the desired result which can be used in the following.
2) The second realization (shown in Fig. 6(c)) makes intensive use of control connections. More precisely, control lines are added to all gates in the realization of the respective then- and else-block. Therewith, the gates in these blocks are only triggered iff the result of the conditional statement (i.e. signal e) is assigned to 1 or 0, respectively. A NOT gate (i.e. a Toffoli gate $t(\emptyset, \{e\})$ without control lines) is thereby applied to flip the value of e so that the gates of the else block can be "controlled" as well.

Having both alternatives, it is up to the designer which one should be used during synthesis. Using the first realization leads to additional circuit lines (particularly in quantum logic a restricted resource). This is not the case in the second realization. But due to the additional control lines, the quantum costs are significantly larger in this solution. Thus, both methods are not completely satisfactory, motivating further research on these higher levels of abstractions.

VI. Conclusions and Outlook

In this paper, we gave a brief overview of the progress that has been made in the design of reversible circuits. We reviewed the basic ideas of several synthesis approaches based on truth table descriptions, exclusive sum of products, decision diagrams, and, finally, a hardware description language. Most of the approaches summarized in this paper are publicly available in RevKit [14]. While this survey shows the significant achievements that have already been realized in this domain, it also manifests some unsolved problems which have to be tackled in future work. For example:

- Synthesis for large functions and synthesis based on hardware description languages, respectively, suffers from a significant number of additional circuit lines. A first approach addressing this issue is already available [41]. However, as e.g. shown in [18], these promising results are still far away from the optimum.
- Synthesis of reversible circuits specified in a hardware description language still is in its infancy. Determining more efficient realizations of the data flow and the control flow remains a subject for future work.
- Quantum cost has been established as a cost metric to evaluate synthesized reversible circuits. But beyond that, also other, more technology-specific constraints should be considered (e.g. transistor cost [42] or nearest-neighbor requirements [43], [44]).
- All synthesis approaches reviewed in this survey considered combinatorial circuits. But in order to realize practical reversible circuits, sequential behavior has to be supported as well. First approaches in this direction have already been proposed (see e.g. [45], [46], [47], [48]) and need to be continued.

Followed by the increasing power of the synthesis methods, also new verification issues will emerge. In particular for complex circuits specified using e.g. hardware description languages, it often cannot be ensured that the design was implemented as intended. Thus, developing appropriate verification methods is a logical next step. Researchers can thereby build on first results achieved for equivalence checking (see e.g. [49], [50], [51]) and even debugging [52].

Furthermore, questions related to test of reversible circuits more and more becomes of interest. Already today, first models and approaches in this area exist (see e.g. [53], [54], [55], [56]). But due to the absence of large physical realizations, it is hard to evaluate the suitability of them. Additionally, existing approaches cover only some possible technologies. With ongoing progress in the development of further (and larger) physical quantum computing or reversible CMOS realizations, new models and approaches are needed to efficiently test them.

Finally, all these methods and approaches have to be combined to an integrated design flow. Even if first approaches towards such a flow are available (see e.g. [57]), this is the long-term goal of any research in the domain of reversible circuit design. The development of a design flow which is comparable to the one for traditional circuit design (that has been developed in the last 25 years) will last further years of research. But the achievements from the recent past provide a good basis for that.

Acknowledgments

We would like to send sincere thanks to Stefan Frehse, Daniel Große, Lisa Jungmann, Sebastian Offermann, and Mathias Soeken for their help in the past years. Many thanks also go to Prof. D. Michael Miller from the University of Victoria, Prof. Gerhard W. Dueck from the University of New Brunswick, and Dr. Mehdi Saeedi from the Amirkabir University of Technology in Tehran for very fruitful collaborations. In this context, we would like to thank the German Academic Exchange Service (DAAD) which enabled the close contact with the groups in Canada. Thanks also go to the German Research Foundation (DFG) which funded parts of our work under the contract number DR 287/20-1.

References

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Dev.*, vol. 5, p. 183, 1961.

[2] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev*, vol. 17, no. 6, pp. 525–532, 1973.

[3] E. F. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3/4, pp. 219–253, 1982.

[4] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, W. de Bakker and J. van Leeuwen, Eds. Springer, 1980, p. 632, technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

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

[6] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Foundations of Computer Science*, pp. 124–134, 1994.

[7] B. Desoete and A. D. Vos, "A reversible carry-look-ahead adder using control gates," *INTEGRATION, the VLSI Jour.*, vol. 33, no. 1-2, pp. 89–104, 2002.

[8] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, p. 883, 2001.

[9] R. Cuykendall and D. R. Andersen, "Reversible optical computing circuits," *Optics Letters*, vol. 12, no. 7, pp. 542–544, 1987.

[10] H. Thapliyal and M. B. Srinivas, "The need of DNA computing: reversible designs of adders and multipliers using Fredkin gate," in *SPIE*, 2005.

[11] R. C. Merkle, "Reversible electronic logic using switches," *Nanotechnology*, vol. 4, pp. 21–40, 1993.

[12] J. P. McGregor and R. B. Lee, "Architectural enhancements for fast subword permutations with repetitions in cryptographic applications," in *Int'l Conf. on Comp. Design*, 2001, pp. 453–461.

[13] Z. Shi and R. B. Lee, "Bit permutation instructions for accelerating software cryptography," in *Int'l Conf. on Application-Specific Systems, Architectures, and Processors*, 2000, pp. 138–148.

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

[15] 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*, vol. 52, pp. 3457–3467, 1995.

[16] 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*, 2011.

[17] D. Maslov and G. W. Dueck, "Reversible cascades with minimal garbage," *IEEE Trans. on CAD*, vol. 23, no. 11, pp. 1497–1509, 2004.

[18] 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.

[19] R. Wille, D. Große, G. Dueck, and R. Drechsler, "Reversible logic synthesis with output permutation," in *VLSI Design*, 2009, pp. 189–194.

[20] D. M. Miller, R. Wille, and G. Dueck, "Synthesizing reversible circuits for irreversible functions," in *EUROMICRO Symp. on Digital System Design*, 2009, pp. 749–756.

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

[22] D. Maslov, G. W. Dueck, and D. M. Miller, "Toffoli network synthesis with templates," *IEEE Trans. on CAD*, vol. 24, no. 6, pp. 807–817, 2005.

[23] P. Kerntopf, "A new heuristic algorithm for reversible logic synthesis," in *Design Automation Conf.*, 2004, pp. 834–837.

[24] P. Gupta, A. Agrawal, and N. K. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 25, no. 11, pp. 2317–2330, 2006.

[25] D. Maslov, G. W. Dueck, and D. M. Miller, "Techniques for the synthesis of reversible Toffoli networks," *ACM Trans. on Design Automation of Electronic Systems*, vol. 12, no. 4, 2007.

[26] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 22, no. 6, pp. 710–722, 2003.

[27] M. Saeedi, M. S. Zamani, M. Sedighi, and Z. Sasanian, "Synthesis of reversible circuit using cycle-based approach," *J. Emerg. Technol. Comput. Syst.*, vol. 6, no. 4, 2010.

[28] W. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis." *IEEE Trans. on CAD*, vol. 25, no. 9, pp. 1652–1663, 2006.

[29] D. Große, R. Wille, G. W. Dueck, and R. Drechsler, "Exact multiple control Toffoli network synthesis with SAT techniques," *IEEE Trans. on CAD*, vol. 28, no. 5, pp. 703–715, 2009.

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

[31] Y. Sanaee and G. W. Dueck, "ESOP-based Toffoli network generation with transformations," in *Int'l Symp. on Multi-Valued Logic*, 2010, pp. 276–281.

[32] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *Design Automation Conf.*, 2009, pp. 270–275.

[33] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[34] R. Wille and R. Drechsler, "Effect of BDD optimization on synthesis of reversible and quantum logic," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 6, pp. 57–70, 2010.

[35] M. Soeken, R. Wille, and R. Drechsler, "Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition," in *Int'l Design & Test Workshop*, 2010.

[36] R. Wille, S. Offermann, and R. Drechsler, "SyReC: A programming language for synthesis of reversible circuits," in *Forum on Specification and Design Languages*, 2010, pp. 184–189.

[37] T. Yokoyama and R. Glück, "A reversible programming language and its invertible self-interpreter," in *Symp. on Partial evaluation and semantics-based program manipulation*, 2007, pp. 144–153.

[38] 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*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.

[39] Y. Takahashi and N. Kunihiro, "A linear-size quantum circuit for addition with no ancillary qubits," *Quantum Information and Computation*, vol. 5, pp. 440–448, 2005.

[40] S. Offermann, R. Wille, G. W. Dueck, and R. Drechsler, "Synthesizing multiplier in reversible logic," in *Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 335–340.

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

[42] M. K. Thomson and R. Glück, "Optimized reversible binary-coded decimal adders," *J. of Systems Architecture*, vol. 54, pp. 697–706, 2008.

[43] M. H. A. Khan, "Cost reduction in nearest neighbour based synthesis of quantum boolean circuits," *Engineering Letters*, vol. 16, pp. 1–5, 2008.

[44] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, 2010.

[45] H. Thapliyal and M. B. Srinivas, "A beginning in the reversible logic synthesis of sequential circuits," in *MAPLD Int'l Conf.*, 2005.

[46] M. Chuang and C. Wang, "Synthesis of reversible sequential elements," in *ASP Design Automation Conf.*, 2007, pp. 420 –425.

[47] N. M. Nayeem, M. A. Hossain, L. Jamal, , and H. Babu, "Efficient design of shift registers using reversible logic," in *Int'l Conf. on Signal Processing Systems*, 2009, pp. 474–478.

[48] H. Himanshu and N. Ranganathan, "Design of reversible sequential circuits optimizing quantum cost, delay, and garbage outputs," *J. Emerg. Technol. Comput. Syst.*, vol. 6, pp. 14:1–14:31, 2010.

[49] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007, pp. 69–74.

[50] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits," *IEICE Transactions*, vol. 91-A, no. 2, pp. 584–594, 2008.

[51] R. Wille, D. Große, D. M. Miller, and R. Drechsler, "Equivalence checking of reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2009, pp. 324–330.

[52] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler, "Debugging of Toffoli networks," in *Design, Automation and Test in Europe*, 2009, pp. 1284–1289.

[53] K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault testing for reversible circuits," *IEEE Trans. on CAD*, vol. 23, no. 8, pp. 1220–1230, 2004.

[54] M. Perkowski, J. Biamonte, and M. Lukac, "Test generation and fault localization for quantum circuits," in *Int'l Symp. on Multi-Valued Logic*, 2005, pp. 62–68.

[55] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes, "A family of logical fault models for reversible circuits," in *Asian Test Symp.*, 2005, pp. 422–427.

[56] H. Zhang, R. Wille, and R. Drechsler, "SAT-based ATPG for reversible circuits," in *Int'l Design & Test Workshop*, 2010.

[57] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. Springer, 2010.