Reducing the Number of Lines in Reversible Circuits

Robert Wille Mathias Soeken Rolf Drechsler
Institute of Computer Science
University of Bremen, Bremen, Germany
{rwille,msoeken,drechsle}@informatik.uni-bremen.de

ABSTRACT

Reversible logic became a promising alternative to traditional circuits because of its applications e.g. in low-power design and quantum computation. As a result, design of reversible circuits attracted great attention in the last years. The number of circuit lines is thereby a major criterion since it e.g. affects the still limited resource of qubits. Nevertheless, all approaches introduced so far for synthesis of complex reversible circuits need a significant amount of additional circuit lines – sometimes orders of magnitude more than the primary inputs.

In this paper, we propose a post-process optimization method that addresses this problem. The general idea is to merge garbage output lines with appropriate constant input lines. To this end, parts of the circuits are re-synthesized. Experimental results show that by applying the proposed approach, the number of circuit lines can be reduced by 17% on average – in the best case by more than 40%. At the same time, the increase in the number of gates and the quantum costs, respectively, can be kept small.

Categories and Subject Descriptors

B.6 [**Hardware**]: Logic Design

General Terms

Algorithms

Keywords

Reversible Logic, Quantum Logic, Optimization

1. INTRODUCTION

The ongoing miniaturization of integrated circuits will reach its limits in the near future. Shrinking transistor sizes and power dissipation are the major barriers in the development of smaller and more powerful circuits. At least when the transistor size approaches the atomic scale, duplication of transistor density (according to Moore's Law) will not be possible any longer. Besides that, power dissipation more and more becomes a crucial issue. Already today, the amount of power dissipated in the form of heat to the surrounding environment of a chip leads to immense challenges in circuit design.

Reversible logic provides an alternative that may overcome many of these problems in the future. For low-power design, reversible logic offers significant advantages since zero power dissipation will only be possible if computation is reversible [8, 2]. Furthermore, quantum computation [13] profits from enhancements in this area, because every quantum circuit is inherently reversible and thus requires reversible descriptions. With the help of quantum circuits many important problems (e.g. factorization) can be solved exponentially faster than by traditional circuits. First physical realizations of both, reversible and quantum circuits, already approved the benefits of these emerging technologies (see e.g. [3, 19]).

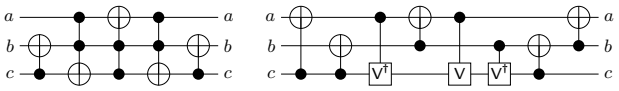
Driven by the promising results, research in the area of reversible logic has been intensified in the last years. As a result, besides verification [20, 22, 25], testing [14, 15, 16], simulation [21, 5], optimization [11], and debugging [24], in particular synthesis of reversible circuits is a main research area. For example, approaches exploiting permutations [17], truth tables [11], positive-polarity Reed-Muller expansions [7], exclusive-or sum-of-products [4], and binary decision diagrams [23] have been introduced.

The number of circuit lines is thereby a major criterion. If the function to be synthesized is reversible, the number of circuit lines can be equal to the number of the primary inputs and primary outputs, respectively. But e.g. to embed irreversible functions, sometimes additional circuit lines (with constant inputs) are unavoidable [10]. Moreover, to guarantee that thereby the number of additionally added circuit lines is minimal, the function to be synthesized must be given in terms of a truth table (or similar descriptions). Approaches that synthesize functions with more than 30 variables (e.g. the ones introduced in [4, 23]) or that apply compositions of basic blocks (e.g. hierarchical synthesis methods like [26]) often need a significant amount of additional circuit lines – sometimes magnitudes more than the primary inputs.

This is a problem since in particular in quantum computation, circuit lines are a highly limited resource (caused by the fact that the number of circuit lines corresponds to the number of qubits). Furthermore, a high number of lines (or qubits, respectively) may decrease the reliability of the resulting system. Thus, keeping the number of circuit lines as small as possible is an important issue. While this was not a problem as long as the synthesis approaches relied on a truth table description (allowing determination of minimal number of lines), with the emergence of synthesis approaches for complex reversible systems, reduction of circuit lines becomes a crucial issue.

In this paper, we propose a post-process optimization method that addresses this problem. Garbage outputs (i.e. circuit lines whose output value is don't care) are thereby exploited. A multi-stage approach is introduced that (1) identifies garbage outputs producing don't cares, (2) re-synthesizes parts of the circuit so that instead of these don't cares concrete constant values are computed, and (3) connects the resulting outputs with appropriate constant inputs. In other words, circuit structures are modified so that they can be merged with constant inputs resulting in a line reduction. For the respective re-synthesis step, existing synthesis methods can be used.

DAC 2010, June 13-18, 2010, Anaheim, California, USA.



(a) Reversible circuit (b) Quantum circuit

Figure 1: Reversible and quantum circuits

Experimental results show that applying this approach, the number of circuit lines can be reduced by 17% on average – in the best case by more than 40%. Furthermore, depending on the used synthesis approach, these line reductions are possible only with a small increase in the number of gates and the quantum costs, respectively. In some cases the costs even can be reduced. In this sense, drawbacks of scalable but line-costly synthesis approaches are minimized.

The remainder of this paper is structured as follows. The next section introduces reversible and quantum logic and gives a brief review of the usage of additional circuit lines in the respective circuits. Section 3 illustrates the general idea of the proposed approach which is afterwards described in Section 4. At the end of the paper, experimental results as well as conclusions are given in Section 5 and Section 6, respectively.

2. BACKGROUND

To keep the paper self-contained, this section briefly reviews the basic concepts of reversible and quantum logic. Afterwards, the usage of additional circuit lines and the resulting consequences are illustrated.

2.1 Reversible and Quantum Logic

A logic function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ over inputs $X = \{x_1, \ldots, x_n\}$ is *reversible* iff (1) its number of inputs is equal to its number of outputs (i.e. $n = m$) and (2) it maps each input pattern to a unique output pattern. That is, reversible functions represent bijections. Reversible circuits are realizations of reversible functions. Fanouts and feedbacks are thereby not allowed [13] so that a reversible circuit G is a cascade of reversible gates g_i, i.e. $G = g_1 \ldots g_d$, where d is the number of gates. Each gate consists of a (possibly empty) set $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ of *control lines* and a set $T = \{x_{j_1}, \ldots, x_{j_l}\} \subset X$ with $C \cap T = \emptyset$ of *target lines*. The gate operation is applied to the target lines iff all control lines meet the required control conditions. Control lines and unconnected lines always pass through the gate unaltered. As an example, the most widely used reversible gate, the *Toffoli* gate [18], has one target line x_j, which is inverted iff all control lines are assigned to 1.

EXAMPLE 1. *Figure 1(a) shows a reversible circuit including Toffoli gates. Control lines are denoted by* $\bullet$*, while the target lines are denoted by* $\oplus$*. This circuit maps e.g. the input* 001 *to the output* 110*.*

Quantum circuits [13] are inherently reversible so that they can easily be derived from every reversible circuit. Each circuit line is thereby represented by a *qubit* – the counterpart of the classical bit. The state of a qubit for two pure logic states can be expressed as $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|0\rangle$ and $|1\rangle$ denote pure logic states 0 and 1, respectively, and α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. Furthermore, each reversible gate is decomposed into a cascade of *quantum gates*. Popular quantum gates are the NOT gate (a single qubit is inverted), the controlled-NOT (CNOT) gate (the target qubit is inverted if the single control qubit is 1), the controlled-V gate (also known as a square root of NOT, since two consecutive V operations are equivalent to an inversion), and the controlled-V+ gate (which performs the inverse operation of the V gate and thus is also a square root of NOT).

Table 1: Truth tables for the AND function

(a) AND function

a	b	$a \wedge b$	
0	0	0	0
0	1	0	1
1	0	0	?
1	1	1	0

(b) AND embedding

0	a	b	$a \wedge b$		
0	**0**	**0**	**0**	0	0
0	**0**	**1**	**0**	0	1
0	**1**	**0**	**0**	1	0
0	**1**	**1**	**1**	1	1
	...			...	

EXAMPLE 2. *Figure 1(b) shows a quantum circuit which realizes the same function as the circuit from Figure 1(a).*

The *costs* of a reversible or quantum circuit are defined by the number of quantum operations needed to realize the circuit. For reversible circuits, the quantum cost depends on the number of control lines of each gate. For example, a Toffoli gate with no or one control line has quantum cost of one, while a Toffoli gate with two control lines has quantum cost of five. Further metrics for the remaining reversible gates can be found e.g. in [1, 9]. For quantum circuits, each gate has quantum cost of one (i.e. the cost of a quantum circuit is equal to the number of gates). Besides quantum cost, also the number of lines is an important metric which is discussed in detail in the next section.

2.2 Additional Lines in Reversible Circuits

Usually, n circuit lines are needed to represent a reversible function $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ in reversible logic. However, if irreversible functions should be synthesized, sometimes further circuit lines with constant inputs must be added [10]. As an example, consider the *AND* function depicted in Table 1(a). This function obviously is 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 inputs and outputs) does not make the function reversible. Then, the first two lines of the truth table can be embedded with respect to reversibility as shown in the rightmost column of Table 1(a), but no unique embedding for the third truth table line is possible any longer. Thus, an additional output (and therewith an additional circuit line with constant input) must be added as depicted in Table 1(b).

More generally, at least $\lceil \log_2 \mu \rceil$ free outputs are required to make an irreversible function reversible, where μ is the maximum number of times an output pattern is repeated in the truth table [10]. The additional lines thereby cause *constant inputs* (i.e. inputs that are set to a fixed value) and *garbage outputs* (i.e. outputs that are don't care for all possible input conditions). In the past, most of the reversible synthesis approaches have been applied to functions with minimal garbage and thus with minimal circuit lines, respectively. However, minimality of additional lines can only be guaranteed, if μ is available. This is the case if the function to be synthesized is given in terms of a truth table (or similar descriptions), but not feasible if the functions e.g. include more than 30 primary inputs. As a result, synthesis approaches for large functions (e.g. [4, 23]) lead to circuits where the number of lines is significantly larger than the optimal value.

A similar issue occurs if circuits are composed using basic blocks. As an example, consider the reversible realization of the *AND* function and the *OR* function as shown in Figure 2(a) and Figure 2(b), respectively. Composing these circuits leads to a realization with two additional circuit

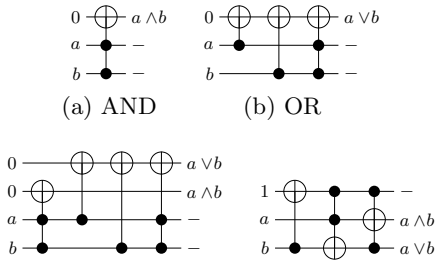


Figure 2: Composition of circuits

lines as in Figure 2(c). In fact, both functions combined can be realized with one additional circuit line only (see Figure 2(d)). However, particularly complex sub-functions (e.g. arithmetic), often can be synthesized by composition only (i.e. by hierarchical synthesis approaches like [26]). In these cases, minimality of the circuit lines cannot be ensured so far.

Overall, while for small functions circuits with minimal lines can easily be synthesized, synthesis of large functions or complex systems often leads to a significant number of additional circuit lines. Since circuit lines are a limited resource (in particular in quantum computation where the number of circuit lines corresponds to the number of qubits), methods that optimize the resulting circuits with respect to the number of lines are needed. Motivated by this, the question considered in this paper is:

> *How can we efficiently reduce the number of lines in reversible circuits?*

In the next sections an approach for this purpose is proposed.

3. GENERAL IDEA

In this section, we present an idea how to reduce the number of lines in large reversible circuits. As discussed above, synthesis approaches ensuring minimality of circuit lines can only handle small functions (because they require e.g. a truth table description). Thus, we consider line reduction as a post-synthesis problem. Our approach thereby exploits a structure often occurring in circuits generated by scalable synthesis approaches (e.g. [4, 23]) or by composed reversible sub-circuits. This is illustrated by the following running example.

EXAMPLE 3. *Consider the circuit $G = g_1 \dots g_{12}$ depicted in Figure 3(a) representing a 3-bit adder that has been created by composing three single (minimal) 1-bit adders. This circuit consists of three additional circuit lines (with constant inputs). Not all of them are necessarily required. Furthermore, there are a couple of garbage outputs whose values are don't care.*

In particular of interest in this circuit is the first usage of a line with a constant input and the last usage of a line with a garbage output. For example, the constant input at line 4 (counted from top) is firstly used by the fifth gate, while at the same time the value of the last line is not needed anymore after the second gate. Since the value of the garbage output doesn't matter (because it is a don't care), this might offer the possibility to merge the line including the constant input with the line including the garbage output. More precisely, if it is possible to modify the circuit so that a garbage output returns a constant value (instead of an arbitrary value), then this constant value can be used in

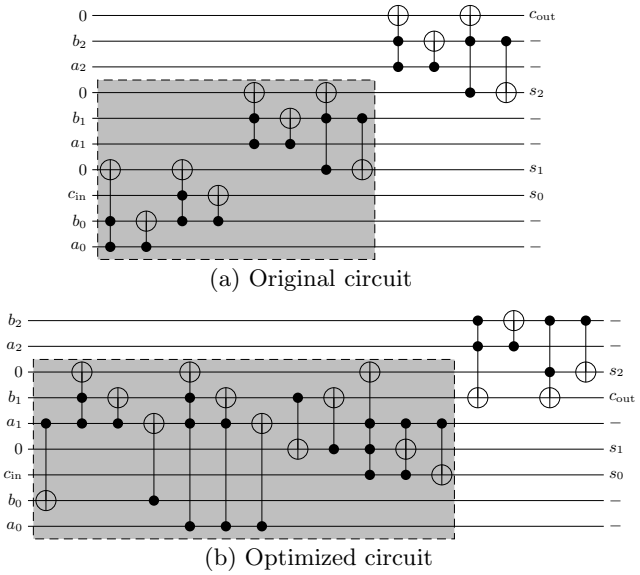


Figure 3: Line reduction in a 3-bit adder circuit

the rest of this circuit. At the same time, a constant input line can be removed. More formally:

PROPOSITION 1. *Without loss of generality, let $G = g_1 \dots g_d$ be a reversible circuit with a constant input at line l_c and a garbage output at line l_g ($l_c \neq l_g$). Furthermore, let g_i be the first gate connected with line l_c (including the constant input) and let g_j with $j < i$ be the last gate connected with line l_g (including the garbage output). If it is possible to modify the sub-circuit $G_1^j = g_1 \dots g_j$ so that line l_g becomes assigned to a constant value, then line l_c can be removed from G. For all gates formerly connected with line l_c, line l_g can be used instead.*

Note that the constant value of the selected line l_c is thereby of no importance. If necessary, the needed value can easily be generated by an additional NOT gate (i.e. a Toffoli gate without any control lines). Furthermore, constant outputs can only be produced if the considered circuit includes additional lines with constant inputs.

EXAMPLE 3 (continued). *Reconsider the adder circuit $G = g_1 \dots g_{12}$ in the running example. The constant input at line 1 is firstly used by gate g_9, while the values of the garbage outputs at line 5, line 6, line 9, and line 10, respectively, are not needed anymore after gate g_8. Since the sub-circuit $G_1^8 = g_1 \dots g_8$ can be modified so that e.g. the garbage output at line 5 becomes assigned to the constant value 0 (see dashed rectangle in Figure 3(b)), line 1 can be removed and the newly created constant value from line 5 can be used instead. The resulting circuit is depicted in Figure 3(b) and consists of 9 instead of 10 lines.*

Note that the respective modification of a sub-circuit is not always possible. For example, consider the constant input at line 4 (firstly used by gate g_5) and the garbage outputs at line 9 and line 10 (not needed anymore after gate g_4). This might offer the possibility to remove one more circuit line. But, the sub-circuit $G_1^4 = g_1 \dots g_4$ cannot be modified accordingly since a realization of the 1-bit addition together with an additional constant output requires more garbage outputs.

Using these observations an algorithm for reducing the number of lines in reversible circuits can be formulated. The

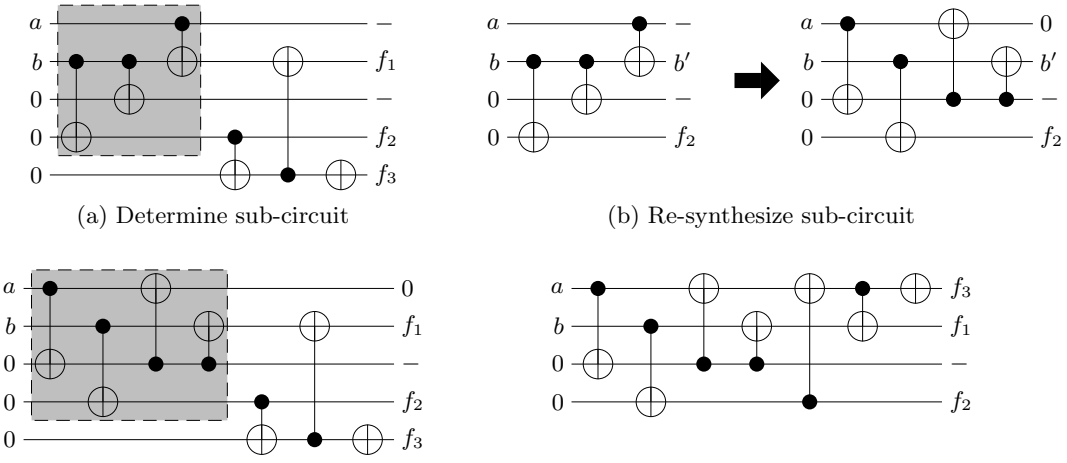


Figure 4: Reducing the number of circuit lines in four steps

next section describes the respective steps in detail. Afterwards, the experiments in Section 5 show that significant reductions for practical relevant circuits can be obtained with this approach.

4. ALGORITHM

Based on the ideas presented in the last section, an algorithm for circuit line reduction is proposed. The respective steps are illustrated by means of an example in Figure 4. At first, an appropriate sub-circuit is determined (a). Afterwards, it is tried to re-synthesize the sub-circuit so that one of the (garbage) outputs returns a constant value (b). If this was successful, the re-synthesized sub-circuit is inserted into the original circuit (c). Finally, the newly created constant output is merged with a line including a constant input (d). The algorithm terminates if no appropriate sub-circuit can be determined anymore. In the following the respective steps are described in detail.

4.1 Determine an Appropriate Sub-circuit

In the considered context, appropriate sub-circuits are characterized by the fact that they include at least one garbage output which can be later used to replace a constant input. Therefore, it is important to know when lines of a circuit are used for the first time and when they are not needed anymore, respectively. This is formalized by the following two functions:

DEFINITION 1. *Let $G = g_1 \dots g_d$ be a reversible circuit. Furthermore, let $l \in \{1, \dots, n\}$ be a line of this circuit. Then, the function* $\mathsf{firstly\_used}(l)$ *returns $i \in \{1, \dots, d\}$ iff g_i is the first gate connected with line l. Accordingly, the function* $\mathsf{lastly\_used}(l)$ *returns $i \in \{1, \dots, d\}$ iff g_i is the last gate connected with line l.*

Using these functions, the flow to determine appropriate sub-circuits can be described as follows:

1. Traverse all circuit lines l_g of the circuit $G = g_1 \dots g_d$ that include a garbage output.

2. Check if line l_g can be merged with another line l_c including a constant input, i.e. if there is a constant input line l_c so that $\mathsf{firstly\_used}(l_\mathrm{c}) > \mathsf{lastly\_used}(l_\mathrm{g})$. If this check fails, continue with the next garbage output line l_g.

3. Check if circuit $G_1^k = g_1 \dots g_k$ with $k = \mathsf{lastly\_used}(l_\mathrm{g})$ can be modified so that line l_g outputs a constant value. If this check fails, continue with the next line l_g in Step 2. Otherwise, G_1^k is an appropriate sub-circuit.

EXAMPLE 4. *Consider the circuit $G = g_1 \dots g_6$ depicted in Figure 4(a). Applying the steps introduced above, the sub-circuit $G_1^3 = g_1 g_2 g_3$ (marked by the dashed rectangle) is determined.*

Note that the order in which the garbage output lines l_g are considered typically has an effect. We consider the lines with the smallest value of $\mathsf{lastly\_used}(l_\mathrm{g})$ first. This is motivated by the fact that $\mathsf{firstly\_used}(l_\mathrm{c}) > \mathsf{lastly\_used}(l_\mathrm{g})$ is a necessary condition which, in particular, becomes true for small values of $\mathsf{lastly\_used}(l_\mathrm{g})$. Besides that, the check in Step 3 is strongly related to the re-synthesis of the sub-circuit which is described next.

4.2 Re-synthesize the Sub-circuit

Given an appropriate sub-circuit G_1^k, the next task is to re-synthesize it so that one garbage output returns a constant value (instead letting it a don't care). Generally, any available synthesis approach can be applied for this purpose. But since we want to reduce the number of circuit lines, approaches that generate additional circuit lines (e.g. [4, 23]) should be avoided. Thus, synthesis methods that require a truth table description (and therewith ensure minimality with respect to circuit lines) are used so far (see Section 1). Consequently, only sub-circuits with a limited number of primary inputs are considered.

To address this issue, not the whole sub-circuit G_1^k is re-synthesized. Instead, a bounded cascade of gates which affects the respective garbage output is considered. More precisely, starting at the output of line l_g, the circuit is traversed towards the inputs of the circuit. Each passed gate as well the lines connected with them are added to the following consideration[1]. The traversal stops, if the number of considered lines reaches a given threshold λ (in our experimental evaluations, it turned out that $\lambda = 6$ is a good choice).

From the resulting cascade, a truth table description is determined. Afterwards, the truth table is modified, i.e. the former garbage output at line l_g is replaced by a constant

[1]In other words, the cone of influence of the garbage output line l_g is considered.

Table 2: Truth tables of the sub-circuit

(a) Orignal

a	b	0	0	–	b'	–	f_1
0	0	0	0	–	0	–	0
				–	–	–	–
	...				...		
0	1	0	0	–	1	–	1
				–	–	–	–
	...				...		
1	0	0	0	–	1	–	0
				–	–	–	–
	...				...		
1	1	0	0	–	0	–	1
				–	–	–	–
	...				...		

(b) After modification

a	b	0	0	0	b'	–	f_1
0	0	0	0	0	0	–	0
				–	–	–	–
	...				...		
0	1	0	0	0	1	–	1
				–	–	–	–
	...				...		
1	0	0	0	0	1	–	0
				–	–	–	–
	...				...		
1	1	0	0	0	0	–	1
				–	–	–	–
	...				...		

output value. It is thereby important that the modification preserves the reversibility of the function. If this is not possible, the sub-circuit is skipped and the next line with a garbage output is considered (see Step 3 from above). Otherwise, the modified truth table can be passed to a synthesis approach.

Note that the modification of the truth table is only possible, if constant values at the primary inputs of the whole circuit are incorporated. Constant inputs restrict the number of possible assignments to the inputs of the considered cascade. This enables a reversible embedding with a constant output.

EXAMPLE 5. *Consider the cascade highlighted by the dashed rectangle in Figure 4(a) which is considered for re-synthesis. Incorporating the constant values at the primary inputs of the whole circuit, only the patterns shown in Table 2(a) have to be considered. The outputs for the remaining patterns are not of interest. This function can be modified so that one of the garbage outputs returns a constant value, while still reversibility of the overall function is preserved (see Table 2(b)). Synthesizing the modified function, the circuit shown on the right-hand side of Figure 4(b) results. This circuit can be used to remove a constant line.*

As shown by the example, re-synthesizing the respective cascades in the described manner might lead to an increase in the number of gates as well as in the quantum costs. This is an expected behavior since circuit lines can be exploited to buffer temporary values (see e.g. [12]). If such lines are removed, additional gates may be required to recompute these values.

4.3 Insert the Sub-circuit and Merge the Lines

If re-synthesis was successful, the last two steps are straight-forward. At first, the considered sub-circuit is replaced by the newly synthesized one. Afterwards, the considered garbage output line l_g is merged with the respective constant input line l_c, i.e. the respective gate connections as well as possible primary outputs are adjusted. Finally, line l_c is removed since it is not needed anymore.

EXAMPLE 6. *Consider the circuit shown in Figure 4. Replacing the highlighted sub-circuit with the re-synthesized one from Example 5, the circuit shown in Figure 4(c) results. Here, line* 1 *and line* 5 *can be merged leading to the circuit depicted in Figure 4(d) where line* 5 *has been removed.*

5. EXPERIMENTAL RESULTS

The proposed approach for line reduction has been implemented in C++ and evaluated using a set of reversible circuits with a large number of constant inputs. As synthesis method for step (b) of the optimization (see Section 4.2), two different approaches have been evaluated, namely

1. an exact synthesis approach (based on the principles of [6] and denoted by *exact synthesis* in the following) that realizes a circuit with minimal number of gates but usually requires a significant amount of run-time and

2. a heuristic synthesis approach (namely the transformation-based method introduced in [11]; in the following denoted by *heuristic synthesis*) that does not ensure minimality but is very efficient regarding run-time.

As benchmarks, reversible circuits from [23] were used. These circuits include a significant number of constant inputs that originated from the synthesis and thus cannot be easily removed. The experiments have been carried out on an Intel Core 2 Duo 2.26 GHz with 3 GB of main memory.

The results of the evaluation are presented in Table 3. The first four columns give the name (*Benchmark*), the number of circuit lines[2] (*Lines*), the gate count (d), and the quantum cost (qc) of the original circuits. In the following columns, the respective values after line reduction as well as the run-time needed for optimization (in CPU seconds) are reported. It is thereby distinguished between results obtained by applying exact synthesis and results obtained by applying heuristic synthesis in Step (b).

As can be seen by the results, the number of lines can be significantly reduced for all considered reversible circuits. On average, the number of lines can be reduced by 17% – in the best case (*spla* with exact synthesis) by more than 40%[3]. As already mentioned in Section 4.2, reducing the circuit lines might lead to an increase in the number of gates as well as in the quantum costs. This is also observable in the results.

In this sense, the differences between the applied synthesis approaches provide interesting insights. While the application of exact synthesis leads to larger run-times (in the worst case more than 3 CPU hours are required), results from the heuristic method are available within minutes. But, the differences in the respective number of gates and the quantum costs, respectively, are significant. If exact synthesis is applied, the increase in number of gates and quantum cost can be kept small – for some circuits (e.g. *cordic* and *spla*) even reductions have been achieved.

6. CONCLUSION

Keeping the number of lines in reversible circuits as small as possible is an important issue, in particular in the design of quantum circuits. However, during synthesis of complex reversible circuits, often additional lines are appended to the circuit. In this paper, we presented a post-optimization approach for circuit line reduction. The general idea is thereby to identify garbage outputs and re-synthesize them so that they can be connected with a constant input. Experiments show that using the proposed approach, the number of lines can be reduced by 17% on average – in the best case by more than 40%. Depending on the approach used for re-synthesis, this might lead to an increase in the number of gates and quantum costs, respectively. However, if exact synthesis is applied for this purpose, the increase remains small – in some cases even a reduction can be observed.

[2]Including both, the number of primary inputs/outputs as well as the number of additional circuit lines.

[3]Note that thereby still the number of primary inputs/outputs are considered which cannot be reduced.

Table 3: Experimental results

	Initial			Line reduction with exact synthesis							Line reduction with heuristic synthesis						
Benchmark	Lines	d	qc	Lines	ΔLines	d	Δd	qc	Δqc	Time	Lines	ΔLines	d	Δd	qc	Δqc	Time
4mod5	7	8	24	6	**(-1)**	10	(2)	26	(2)	0.00	6	**(-1)**	10	(2)	26	(2)	0.00
mini-alu	10	19	59	9	**(-1)**	19	(0)	88	(29)	1.96	9	**(-1)**	95	(76)	670	(611)	0.02
rd53	13	34	98	12	**(-1)**	35	(1)	99	(1)	0.12	12	**(-1)**	50	(16)	227	(129)	0.01
sym6	14	28	92	11	**(-3)**	28	(0)	92	(0)	41.11	11	**(-3)**	177	(149)	1340	(1248)	0.05
9sym	27	61	205	24	**(-3)**	61	(0)	209	(4)	3489.19	22	**(-5)**	362	(301)	2845	(2640)	0.51
sym9	27	61	205	24	**(-3)**	61	(0)	209	(4)	3485.33	22	**(-5)**	362	(301)	2845	(2640)	0.54
hwb5	28	88	276	26	**(-2)**	89	(1)	342	(66)	3913.73	25	**(-3)**	146	(58)	915	(639)	0.19
mod5adder	32	96	292	25	**(-7)**	96	(0)	381	(89)	1899.34	25	**(-7)**	223	(127)	1379	(1087)	487.93
rd84	34	102	302	25	**(-9)**	105	(3)	362	(60)	3056.40	25	**(-9)**	424	(322)	3416	(3114)	1.84
cycle10_2	39	71	195	31	**(-8)**	76	(5)	253	(58)	1290.73	30	**(-9)**	585	(514)	3867	(3672)	1.35
ham15	45	152	308	37	**(-8)**	164	(12)	365	(57)	2117.42	37	**(-8)**	677	(525)	4652	(4344)	3.82
hwb6	46	159	507	41	**(-5)**	172	(13)	572	(65)	2141.84	41	**(-5)**	485	(326)	3293	(2786)	1.38
cordic	52	100	324	40	**(-12)**	80	(-20)	264	(-60)	6280.85	39	**(-13)**	805	(705)	5909	(5585)	5.77
hwb7	73	281	909	66	**(-7)**	288	(7)	1010	(101)	1204.57	65	**(-8)**	586	(305)	4385	(3476)	5.12
bw	87	286	922	71	**(-16)**	292	(6)	1167	(245)	11255.00	72	**(-15)**	467	(181)	2623	(1701)	1.76
hwb9	170	699	2275	152	**(-18)**	718	(19)	2545	(270)	4656.58	152	**(-18)**	1354	(655)	8437	(6162)	254.10
ex5p	206	625	1821	165	**(-41)**	615	(-10)	2250	(429)	2625.17	171	**(-35)**	909	(284)	5651	(3830)	108.52
spla	489	1669	5885	267	**(-222)**	1007	(-662)	3805	(-2080)	342.75	329	**(-160)**	2945	(1276)	22392	(16507)	894.81

Acknowledgment

This work was supported by the German Research Foundation (DFG) (DR 287/20-1).

7. REFERENCES

[1] 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.

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

[3] B. Desoete and A. D. Vos. A reversible carry-look-ahead adder using control gates. *INTEGRATION, the VLSI Jour.*, 33(1-2):89–104, 2002.

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

[5] D. Goodman, M. A. Thornton, D. Y. Feinstein, and D. M. Miller. Quantum logic circuit simulation based on the QMDD data structure. In *Int'l Reed-Muller Workshop*, 2007.

[6] 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.

[7] 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.

[8] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183, 1961.

[9] D. Maslov and G. W. Dueck. Improved quantum cost for n-bit Toffoli gates. *IEE ELECTRONICS LETTERS*, 39:1790, 2004.

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

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

[12] D. M. Miller, R. Wille, and R. Drechsler. Reducing reversible circuit cost by adding lines. In *Int'l Symp. on Multi-Valued Logic*, 2010.

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

[14] K. N. Patel, J. P. Hayes, and I. L. Markov. Fault testing for reversible circuits. *IEEE Trans. on CAD*, 23(8):1220–1230, 2004.

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

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

[17] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 22(6):710–722, 2003.

[18] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, page 632. Springer, 1980. Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[19] 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*, 414:883, 2001.

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

[21] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes. Gate-level simulation of quantum circuits. In *ASP Design Automation Conf.*, pages 295–301, 2003.

[22] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo. An XQDD-based verification method for quantum circuits. *IEICE Transactions*, 91-A(2):584–594, 2008.

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

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

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

[26] R. Wille, S. Offermann, and R. Drechsler. SyReC: A programming language for synthesis of reversible circuits. 2010. Tech. Rep., University of Bremen, 2010.