Determining the Minimal Number of Lines for Large Reversible Circuits

Robert Wille Oliver Keszöcze Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{rwille, keszocze, drechsle}@informatik.uni-bremen.de

***Abstract*—Synthesis of reversible circuits is an active research area motivated by its applications e.g. in quantum computation or low-power design. The number of used circuit lines is thereby a crucial criterion. In this paper, we introduce several methods (including a theoretical upper bound) for the efficient computation or at least approximation of the minimal number of lines needed to realize a given function in reversible logic. While the proposed exact approach requires a significant amount of run-time (exponential in the worst case), the heuristic methods lead to very precise approximations in very short run-time. Using this, it can be shown that current synthesis approaches for large functions are still far away from producing optimal circuits with respect to the number of lines.**

I. Introduction

Reversible circuits realize functions with the same number of inputs and outputs, whereby each input pattern is mapped to a unique output pattern (i.e. bijections are realized). While conventional (irreversible) circuit technologies more and more suffer e.g. from shrinking transistor sizes and power dissipation, reversible circuits offer some promising applications, e.g. in quantum computation [1] and low-power design [2].

Accordingly, how to efficiently synthesize reversible circuits has been studied intensely in the last years. Starting with approaches based on truth table-like descriptions of the function to be synthesized (see e.g. [3], [4], [5]), today approaches are available, which can handle more compact function descriptions [6] or even hardware description languages [7]. The number of used circuit lines is thereby a crucial criterion. In particular, in the domain of quantum computation, circuit lines are represented by so called qubits – a limited resource. Furthermore, the number of lines has a close relation to the reliability of the circuit. Thus, it is well-accepted that the number of circuit lines in reversible circuits should be kept small.

For small functions to be synthesized (i.e. for functions given in terms of a truth table), keeping the number of circuit lines small is easily possible [8]. In fact, most of the respective approaches (e.g. [3], [4], [5]) generate circuits with a minimal number of circuit lines. In contrast, no approach is known so far, which synthesizes reversible circuits with minimal number of lines for larger functions (i.e. for functions, which cannot be represented in terms of a truth table any longer). Instead, only heuristic approaches (e.g. [6]) are available. But, it is unclear how far away the resulting circuits are from the (theoretical) optimum.

In this paper, the question how to determine the minimal number of lines in a reversible circuit is addressed. Therefore, we propose methods for the efficient computation or the approximation of the minimal number of lines needed to realize a given function in reversible logic. More precisely, a theoretical upper bound is presented followed by a heuristic approach. Both methods approximate the minimal number of circuit lines, but already give a very close indication of the actual minimum. Afterwards, an exact approach is presented. In contrast to the approximations, this requires significantly more run-time (in the worst case exponential), but leads to the exact results.

978-3-9810801-7-9/DATE11/©2011 EDAA

Experiments show that with the proposed approaches, the minimal number of lines for some functions can be computed for the first time. But, due to the exponential worst case behavior, the exact method reaches its limits quite early. However, the additionally proposed theoretical bound and the heuristic approach provide a very precise approximation in very short run-time. Using this, it can be shown that current synthesis approaches for large functions are still far away from the optimum with respect to the number of lines.

The remainder of this paper is organized as follows: Section II provides the background on reversible functions and the number of circuit lines in the corresponding circuits. Then, the heuristic approaches (including the theoretical upper bound) and the exact approach are proposed in Section III and Section IV, respectively. Finally, the experiments are discussed in Section V and conclusions are drawn in Section VI.

II. Minimal Number of Lines in Reversible Circuits

In this section, reversible logic is introduced and the addressed problem is motivated, respectively. The introduction is thereby kept brief. For a more detailed treatement, we refer to the respective literature (e.g. [1]).

A Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ is *reversible* iff

- its number of inputs is equal to the number of outputs (i.e. $n = m$) and
- it maps each input pattern to a unique output pattern.

Reversible functions are realized by reversible circuits. A *reversible circuit* G is a cascade of reversible gates, where fanout and feedback are not directly allowed [1]. In the literature, reversible circuits composed of *Toffoli gates* are frequently used. Each variable of the function f is thereby represented by a *circuit line*, i.e. a signal through the whole cascade structure on which the respective computation is performed.

In order to synthesize compact reversible circuits, the number of used circuit lines is crucial – in particular for applications in the domain of quantum computation. Here, each circuit line corresponds to a qubit – a limited resource. Besides that, a large number of lines may decrease the reliability of the resulting system. For this, the number of circuit lines (or qubits, respectively) has to be kept as small as possible.

The following observations are thereby applied: If the function $f : \mathbb{B}^n \to \mathbb{B}^n$ to be synthesized is reversible, then obviously only n circuit lines are needed. In contrast, if irreversible functions are synthesized, additional lines might be required as the following example shows.

TABLE I
BOOLEAN FUNCTIONS

(a) Irrev. (Adder)

x_1	x_2	x_3	f_1	f_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) Incompl. embedding

x_1	x_2	x_3	f_1	f_2	
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

(c) Complete embedding

0	x_1	x_2	x_3	f_1	f_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
				...			

Example 1. *Consider the adder function from Table I(a). The adder 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) would not make the function reversible. Then, without loss of generality, the first four lines of the truth table can be embedded with respect to reversibility as shown in the rightmost column of Table I(b). However, since* $f_1 = 0$ *and* $f_2 = 1$ *already appeared two times (marked bold), no unique embedding for the fifth line is possible any longer. The same also holds for the lines shown in italic. Hence, additional outputs are needed. This may lead to more circuit lines.*

In general, at least $\lceil \log_2(\mu) \rceil$ additional outputs (also called *garbage outputs*) are required to make an irreversible function reversible [8], whereby μ is the maximal number of times an output pattern is repeated in the truth table. Thus, to realize an irreversible function $f : \mathbb{B}^n \to \mathbb{B}^m$, at least $m + \lceil \log_2(\mu) \rceil$ circuit lines are required. The resulting circuit then would have n primary inputs, $m + \lceil \log_2(\mu) \rceil - n$ constant inputs, m primary outputs, and $\lceil \log_2(\mu) \rceil$ garbage outputs[1]. The values of the constant inputs and the garbage outputs can thereby be chosen arbitrarily as long as they ensure a unique input/output mapping.

Example 1 (cont.). *In case of the adder function,* $\lceil \log_2(3) \rceil = 2$ *additional outputs are required, since at most three output patterns are repeated, i.e.* $\mu = 3$*. Thus, beside the* 3 *primary inputs and* 2 *primary outputs, a reversible realization of an adder must consist of at least* 1 *constant input and* 2 *garbage outputs. A possible assignment to the newly added values is depicted in Table I(c) (the original adder function is highlighted in bold).*

However, all approaches proposed in the past in order to determine μ consider the truth table of the function to be synthesized. Due to the exponential blow-up of truth table specifications, this is affordable for small functions only. In contrast, minimality with respect to the number of circuit lines has not been shown for large functions so far. Moreover, all existing synthesis approaches that can handle such large functions (e.g. [6]), make extensive use of additional circuit lines. First approaches exist aiming at the reduction of this amount [9]. But, how far away these results are from the theoretical optimum is an open problem. This is addressed in this work.

More precisely, theoretical results and practical methods for this purpose are introduced. The aim is thereby to determine or to approximate the value μ, respectively, without considering the whole truth table. Heuristic results (i.e. approximations of the minimal number of circuit lines) as well as exact results (i.e. minimal values) are obtained. Having these results, it can be shown that current synthesis approaches for large functions lead to circuits whose number of lines is far away from the optimum.

[1]Note that if the number of primary inputs n is larger than $m + \lceil \log_2(\mu) \rceil$, of course at least n circuit lines are needed to realize the function.

III. Heuristic Computation

In this section, heuristic methods are presented, i.e. approaches that approximate the minimal number of circuit lines very fast, but do not guarantee minimality. A theoretical result is presented first, which can be exploited for this purpose. Afterwards, an alternative is introduced, which works on a two-level description of the given function.

A. Theoretical Consideration

Having a function to be synthesized available, an upper bound for the number of required circuit lines can be determined using the following lemma.

Lemma 1 (Upper Bound)**.** *Given a function* $f : \mathbb{B}^n \to \mathbb{B}^m$*. To realize* f *as a reversible circuit, at most* $m + n$ *lines are needed.*

Proof: The minimal number of lines needed to realize the function f *as a reversible circuit is* $m + \lceil \log_2(\mu) \rceil$*, where* μ *is the maximal number of times an output pattern is repeated. In the worst case, the maximal value of* μ *is* 2^n*. This is the case, if* f *is a constant function, i.e. all* 2^n *possible input pattern map to the same output pattern. This leads to* $m + \lceil \log_2(2^n) \rceil = m + n$*.* ■

Example 2. *Again, consider the adder function shown in Table I(a). This function has* $n = 3$ *primary inputs and* $m = 2$ *primary outputs. Thus, in order to realize this adder at most* $3 + 2 = 5$ *circuit lines are needed.*

As shown by the example, Lemma 1 only is an approximation (more precisely, an upper bound) of the minimum. The exact minimal number of circuit lines needed for the adder function is 4 instead of 5 (see Example 1 and Table I(c), respectively). However, as also confirmed by the experiments in Section V, this upper bound already gives a very close approximation of the exact value. Additionally, the bound is easy to determine, since only the number of primary inputs and the number of primary outputs have to be summed up.

B. Exploiting Two-Level Descriptions

In order to avoid the (exponentially large) truth table, Boolean functions are often represented by two-level descriptions, like *Sum of Products* (SoPs). Here, the function is defined by a disjunction of conjunctions, which allows a more compact specification in many cases.

As an example, consider the function given in the table in Fig. 1. The column on the left-hand side gives the respective conjunctions of the primary inputs, 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 conjuntion. The right-hand side gives the respective primary output patterns. The disjunction of all rows leads to the overall function. Thus, instead of $2^5 = 32$ truth table lines, the function can be represented by 6 lines only. This kind of description is frequently used in logic synthesis. In particular, the PLA format (used e.g. by Espresso [10]) relies on that.

Having such a description, the minimal number of lines needed to realize the function as a reversible circuit can be approximated as illustrated by the following example.

x_1	x_2	x_3	x_4	x_5	f_1	f_2	f_3		
1	-	-	0	-	1	0	0	8	
0	0	-	-	-	0	1	0	8	
1	1	-	-	1	0	0	1	4	4+8=12
-	1	0	-	-	0	0	1	8	
1	0	-	1	-	1	0	1	4	
1	1	-	1	0	1	0	1	2	4+2=6

Fig. 1. Two-level description of a Boolean function

Example 3. *Consider the function given in Fig. 1. The maximal number μ of times an output pattern is repeated can be approximated from this two-level description. For example, it can be seen that the conjunction of x_1, x_2, and x_5 (represented by `11--1`) lead to the output pattern* 001*. Since x_2 and x_3 are not part of the conjunction, this results in $2^2 = 4$ input patterns. Additionally, also the conjunction of x_2 and $\overline{x}_3$ (represented by `-10--`) lead to* 001*, resulting in further $2^3 = 8$ input patterns for this case. Thus, about $4 + 8 = 12$ input patterns lead to* 001 *– more than any other output pattern in the SoP-description (see right-hand side of Fig. 1). As a result, μ approximately is* 12*, i.e. about $3 + \lceil \log_2(12) \rceil = 3 + 4 = 7$ lines are needed to realize this function as a reversible circuit.*

The determined value is still an approximation, since overlaps of the respective conjunctions are not considered. For example, the two conjunctions discussed in Example 3 share some equal input patterns, i.e. the determined number of 12 occurrences of the output pattern 001 is an over-approximation. Furthermore, in the case of an overlap, the output values may be subject to a disjunction leading to different patterns and therewith a different value for μ. Excluding all overlaps is a (computational) hard task, which in the worst case would lead to the exponential specification in terms of a truth table. This is discussed in the next section. However, many SoP specifications have very few overlaps. Thus, as the upper bound introduced in the previous section, also this approximation is quite close to the exact minimum in many cases.

IV. Exact Computation

As discussed in the previous section, two-level descriptions provide a good starting point to determine the maximal number μ of times an output pattern is repeated. However, in order to achieve exact results (in contrast to an approximation), all overlaps in the given description have to be removed.

One way to do so is shown by the algorithm given in Fig. 2. The general flow is as follows: The original set OC of all conjunctions given by the SoP-description is traversed. The first conjunction c_{first} of OC is thereby assumed to be overlapping-free and, thus, added to a new set NC of overlapping-free conjunctions (Line 1/2). Then, for all remaining conjunctions c (Line 3), the following steps are performed:

- The conjunction c is compared to all conjunctions c' already added to NC (Line 4).
- If c and c' have no overlaps (Line 5), then *canBeAdded* is assigned to true (Line 6). If this holds for all remaining conjunctions c' of NC, then c is overlapping-free and, thus, can simply be added to NC (Line 13/14).
- If in contrast, c and c' do have overlaps (Line 7), c cannot simply be added to NC (i.e. *canBeAdded* is assigned to false; Line 8). Instead, the respective conjunctions are subject to a special treatment (Lines 9-11). More precisely, conjunctions covering the input patterns of c but excluding the input patterns covered by c' are added to OC (this is denoted by $c - c'$ in Fig. 2). These conjunctions will be considered later in the following iterations. In contrast, conjunctions covering all remaining input patterns are added to NL. These conjunctions are replacing c'. Note that c and c' may have different outputs. In this case, a bit-wise OR is performed on the output patterns of the overlapping conjunctions (denoted by $(c \wedge c')_{OR}$ in Fig. 2) in order to stay conform with the SoP definition.

Input : OC (set of all conjunctions given by the SoP)
Output: NC (overlapping-free set of all conjunctions)

1 $OC \leftarrow OC \setminus \{c_{first}\}$, whereby $c_{first} \in OC$
2 $NC \leftarrow \{c_{first}\}$
3 **foreach** $c \in OC$ **do**
4 **foreach** $c' \in NC$ **do**
5 **if** $c \wedge c' = 0$ **then**
  `// c and c' do not overlap`
6 canBeAdded $\leftarrow$ **true**
7 **else**
  `// c and c' overlap`
8 canBeAdded $\leftarrow$ **false**
9 $OC \leftarrow OC \cup \{c - c'\}$
10 $NC \leftarrow (NC \setminus \{c'\})$
11 $NC \leftarrow NC \cup \{c' - c\} \cup \{(c \wedge c')_{OR}\}$
12 **break**
13 **if** *canBeAdded* **then**
14 $NC \leftarrow NC \cup \{c\}$
15 **return** NC

Fig. 2. Removing overlaps in SoP descriptions

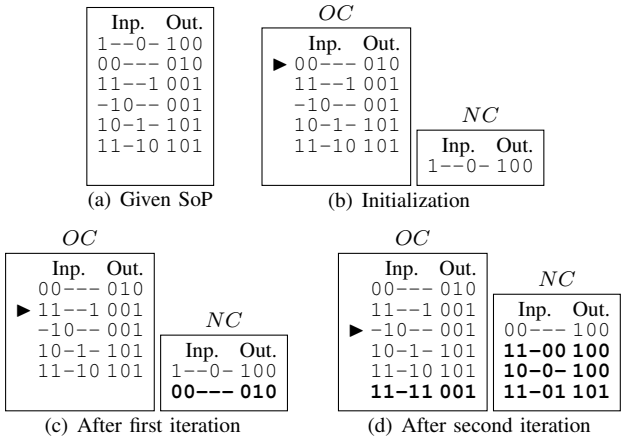

Fig. 3. Application of the exact algorithm

Example 4. *The described algorithm is applied to the SoP specification shown in Fig. 3(a). According to the first two lines of the algorithm, the sets OC and NC are initialized as depicted in Fig. 3(b). Afterwards, the set OC is traversed, starting with the conjunction c represented by `00---`. Since c has no overlap with any conjunction in NC, c is added to NC (highlighted bold in Fig. 3(c)). Next, the conjunction c represented by `11--1` is considered. Here, an overlap with the conjunction $c' \in NC$ represented by `1--0-` is identified. As a result, a conjunction covering the input patterns of c but excluding the input patterns covered by c' (i.e. `11-11`) is added to OC. In contrast, conjunctions covering all remaining input patterns (i.e. `11-00`, `10-0-`, and `11-01`) are added to NL. Since the conjunction `11-01` represents thereby the actual overlap, the respective output patterns are ORed in this*

TABLE II
EXPERIMENTAL RESULTS

Function			Number of Circuit Lines				
	$\|PI\|$	$\|PO\|$	BDD [6]	Lemma 1	Heur. Alg.	Exact Alg.	
apex2	39	3	498	42	43	–	(TO)
apex5	117	88	1147	205	207	–	(TO)
cordic	23	2	52	25	28	–	(TO)
cps	24	109	930	133	135	–	(TO)
e64	65	65	195	130	129	129	(314.51s)
ex5p	8	63	206	71	68	68	(0.07s)
pdc	16	40	619	56	61	55	(1.26s)
seq	41	35	1617	76	76	–	(TO)
spla	16	46	489	62	65	61	(1.51s)
xor5	5	1	6	6	5	5	(0.07s)

case. The resulting assignments of the two sets are depicted in Fig. 3(d). Analogously, this process continues until the set OC has completely been traversed.

Note that this approach can analogously be applied for other two-level descriptions. For example, if *Exclusive Sum of Products* (ESoPs) are used, only the respective treatment of the output patterns has to be adjusted (instead of ORing, the outputs have to be EXORed). Furthermore, note that the conjunctions leading to the output pattern `0...0` (i.e. all outputs assigned to 0), are often not explicitly given in a two-level description. However, the number of times this particular output is generated can easily be concluded since the occurrences of all other output patterns are available.

Applying this algorithm, an overlapping-free set of all conjunctions is computed. From this, the maximal value μ of times an output pattern is repeated can be extracted. In the worst case, the whole truth table will thereby be unfolded. That is, in this case the approach remains exponential. However, the presented algorithm tries to avoid this by traversing the respective conjunctions and terminating as soon as all overlaps have been removed. In doing so, for certain functions the minimal number of lines can be determined, which would not be possible with a truth table description.

V. EXPERIMENTAL EVALUATION

Using the approaches introduced in the previous sections, the number of lines for a set of functions has been approximated or explicitly determined, respectively. The proposed algorithms have been implemented in C++ on top of RevKit [11] and evaluated using an AMD Dual-Core 2.8 GHz with 32 GB of memory. The timeout (denoted by TO) was set to 5000 CPU seconds. For comparison, the results obtained by the BDD-based synthesis approach from [6] are considered. As benchmarks commonly used functions taken from the LGSynth package have been used.

The resulting data is summarized in Table II. The first columns provide the name of the function, the number of primary inputs (denoted by $|PI|$), and the number of primary outputs (denoted by $|PO|$). The next column gives the number of lines of the circuits generated by the BDD-based approach. Afterwards, the heuristic results obtained by both, Lemma 1 and the approach introduced in Section III-B, are reported (denoted by Lemma 1 and Heur. Alg., respectively). Finally, the exact results obtained by the algorithm introduced in Section IV are given (denoted by Exact Alg.). All results have been obtained in negligible run-time, except for the exact approaches, where the run-times (in CPU seconds) are provided in brackets.

So far, synthesis of reversible circuits for large functions is only possible with hierarchical approaches, like the BDD-based approach. As can be seen, this leads to a significant amount of additional circuit lines. For example, the BDD-circuit for the function *pdc* with its 16 primary inputs and 40 primary outputs includes more than 600 circuit lines. In contrast, as shown by the exact computation, not more than 55 lines are needed to realize this function. Hence, there is still a huge gap between the results obtained by current synthesis approaches for large functions and the actual minimum of circuit lines. This clearly emphasizes the need for efficient algorithms determining or at least approximating the minimal number of lines in reversible circuits.

As expected, the proposed exact approach reaches thereby its limits due to long run-times quite early. Nevertheless, for some cases the minimum can be achieved within the given timeout (e.g. for the first time, the minimum was obtained for the function *e64* with its 65 primary inputs). However, for the majority of benchmarks, this is not possible due to the exponential worst case behavior. But, as can be seen e.g. for *e64*, *ex5p*, *pdc*, *spla*, and *xor5*, the heuristic results (shown in Column Lemma 1 and Heur. Alg.) provide a very good approximation. In fact, the heuristic results differ by at most a value of 6 from the exact result. Considering that these heuristic results are determined in nearly no run-time, this is a very good approximation, which in particular can be used to evaluate state-of-the-art synthesis approaches for large functions. For example, with the heuristic result, it can be shown that the number of circuit lines of the BDD-circuit for the function *apex2* is still far away from the upper bound obtained by Lemma 1.

VI. CONCLUSION

Synthesis of reversible circuits is an active research area for emerging technologies with promising applications. The number of circuit lines is thereby a crucial criterion, since they e.g. represent the number of qubits in quantum applications or have a close relation to the reliability of the circuit, respectively. In this paper, we presented several methods (including a theoretical upper bound) for efficient computation or at least approximation of the minimal number of lines needed to realize a given function in reversible logic. The results showed that current synthesis approaches for large function are still far away from the optimum with respect to the number of lines. This clearly motivates further research on improving synthesis of reversible circuits for large functions.

ACKNOWLEDGMENT

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

REFERENCES

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

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

[3] 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.

[4] 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.

[5] 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.

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

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

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

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

[10] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

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