Please note: The methods introduced in this paper are availabe at www.revkit.org.

BDD-based Synthesis of Reversible Logic for Large Functions

Robert Wille
Institute of Computer Science
University of Bremen
28357 Bremen, Germany
rwille@informatik.uni-bremen.de

Rolf Drechsler
Institute of Computer Science
University of Bremen
28357 Bremen, Germany
drechsler@uni-bremen.de

ABSTRACT

Reversible logic is the basis for several emerging technologies such as quantum computing, optical computing, or DNA computing and has further applications in domains like low-power design and nanotechnologies. However, current methods for the synthesis of reversible logic are limited, i.e. they are applicable to relatively small functions only. In this paper, we propose a synthesis approach, that can cope with Boolean functions containing more than a hundred of variables. We present a technique to derive reversible circuits for a function given by a *Binary Decision Diagram* (BDD). The circuit is obtained using an algorithm with linear worst case behavior regarding run-time and space requirements. Furthermore, the size of the resulting circuit is bounded by the BDD size. This allows to transfer theoretical results known from BDDs to reversible circuits. Experiments show better results (with respect to the circuit cost) and a significantly better scalability in comparison to previous synthesis approaches.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Reversible Logic, Quantum Logic, Synthesis, Decision Diagrams

1. INTRODUCTION

Reversible logic [1, 2, 3] realizes n-input n-output functions that map each possible input vector to a unique output vector (i.e. bijections). Although reversible logic significantly differs from traditional (irreversible) logic (e.g. fan-out and feedback are not allowed), it has become an intensely studied research area in recent years. In particular, this is caused by the fact that reversible logic is the basis for several emerging technologies, while traditional methods suffer from the increasing miniaturization and the exponential growth of the number of transistors in integrated circuits.

In fact, reversible logic can help to face the enormous challenges in the development of future computing machines: While for example the increasing power consumption of electronic devices becomes a serious problem in current technologies, energy dissipation is reduced or even eliminated if computation becomes information-lossless [1]. This holds for reversible logic, since data is bijectivly transformed without loosing any of the original information. Furthermore, reversible computation is the basis for quantum computing [4]. In this domain it has been shown that important problems such as factorization can be solved exponentially faster than by classical methods. Further applications of reversible logic can be found in the domain of optical computing [5], DNA computing [2], and nanotechnologies [6].

However, currently the synthesis of reversible logic or quantum circuits, respectively, is limited. Exact (see e.g. [7, 8]) as well as heuristic (see e.g. [9, 10, 11, 12, 13, 14]) methods have been proposed. But both are applicable only for relatively small functions. Exact approaches reach their limits with functions containing more than 6 variables [8] while heuristic methods are able to synthesize functions with at most 30 variables [13]. Moreover, often a significant amount of run-time is needed to achieve these results.

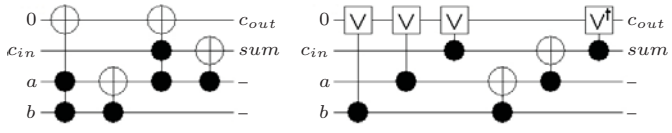
These limitations are caused by the underlying techniques. The existing synthesis approaches often rely on truth tables (or similar descriptions like permutations) of the function to be synthesized (e.g. in [9, 10]). But even if more compact data-structures like BDDs [11], positive-polarity Reed-Muller expansion [13], or Reed-Muller spectra [14] are used, the same limitations can be observed since all these approaches apply similar strategies (namely selecting reversible gates so that the choosen function representation becomes the identity).

In this work we introduce a synthesis method that can cope with significantly larger functions. The basic idea is as follows: First, for the function to be synthesized a BDD [15] is built. This can be efficiently done for large functions using existing well-developed techniques. Then, each node of the BDD is substituted by a cascade of reversible gates. Since BDDs may include shared nodes causing fan-outs (which are not allowed in reversible logic), this may require additional circuit lines.

As a result, circuits composed of Toffoli [3] or elementary quantum gates [4], respectively, are obtained in linear time and with memory linear to the size of the BDD. Moreover, the size of the resulting circuit is bounded by the BDD, so that theoretical results known from BDDs (see e.g. [16, 17]) can be transferred to reversible circuits. Our experiments show significant improvements (with respect to the resulting circuit cost as well as to the run-time) in comparison to previous approaches. Furthermore, for the first time large functions with more than a hundred of variables can be synthesized at very low run-time.

The remainder of the paper is structured as follows: Section 2 provides the basics of reversible logic and BDDs. Afterwards, in Section 3 the synthesis approach is described in detail. Section 4 briefly reviews some of the already known theoretical results from reversible logic synthesis and introduces bounds which follow from the new synthesis approach. Finally, in Section 5 experimental results are given and the paper is concluded in Section 6.

DAC 2009, July 26 - 31, 2009, San Francisco, California, USA.



(a) With Toffoli gates (b) With elementary quantum gates

Figure 1: Two circuits realizing a full adder

2. PRELIMINARIES

To keep the paper self-contained this section briefly reviews the concepts of reversible and quantum logic. We also describe the basics of BDDs which are used as the main data-structure in our synthesis approach.

2.1 Reversible and Quantum Logic

A logic function is reversible if it maps each input assignment to a unique output assignment. Such a function must have the same number of input and output variables $X := \{x_1, \ldots, x_n\}$. Since fanout and feedback are not allowed in reversible logic, a circuit realizing a reversible function is a cascade of reversible gates. A *reversible gate* has the form $g(C, T)$, where $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ is the set of control lines and $T = \{x_{j_1}, \ldots, x_{j_l}\} \subset X$ with $C \cap T = \emptyset$ is the set of target lines. C may be empty. 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.

In the literature, several types of reversible gates have been introduced. Besides the Fredkin gate [18] and the Peres gate [19]), *(multiple controlled) Toffoli* gates [3] are widely used. Each Toffoli gate has one target line x_j, which is inverted iff all control lines are assigned to 1. That is, a multiple controlled Toffoli gate maps $(x_1, \ldots, x_j, \ldots, x_n)$ to $(x_1, \ldots, x_{i_1} x_{i_2} \cdots x_{i_k} \oplus x_j, \ldots, x_n)$.

The *cost* of a reversible circuit is defined either by the number of gates or by so called *quantum cost* [20, 21]. The latter can be derived by substituting the reversible gates of a circuit by a cascade of *elementary quantum* gates [4]. Elementary quantum gates realize quantum circuits that are inherently reversible and manipulate qubits rather than pure logic values. 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 0 and 1, respectively, and α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. The most used elementary 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).

EXAMPLE 1. *Figure 1(a) shows a Toffoli gate realization of a full adder. A circuit realizing the same function by elementary quantum gates is depicted in Figure 1(b).*

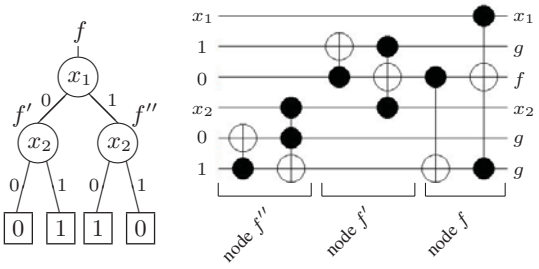
Since quantum circuits are reversible, to realize a non-reversible function (e.g. an n-input m-output function with $n > m$) it must be embedded into a reversible one [22]. Therefore, it is often necessary to add *constant inputs* and *garbage outputs*. The garbage outputs are by definition don't cares and can be left unspecified.

2.2 Binary Decision Diagrams

A Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ can be represented by a *Binary Decision Diagram* (BDD) [15]. A BDD is a directed acyclic graph $G = (V, E)$ where a Shannon decomposition

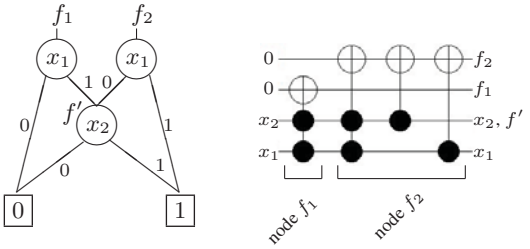
$$f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1} \quad (1 \leq i \leq n)$$

is carried out in each node $v \in V$. In the following the node representing $f_{x_i=0}$ ($f_{x_i=1}$) is denoted by $low(v)$ ($high(v)$) while x_i is called the *select variable*. The *size* k of a BDD is defined by the number of non-terminal nodes.



(a) BDD (b) Resulting circuit

Figure 2: BDD and Toffoli circuit for $f = x_1 \oplus x_2$



(a) BDD (b) Resulting circuit

Figure 3: Synthesizing $f_1 = x_1 \wedge x2$ and $f_2 = x_1 \vee x_2$

The size of a BDD can be significantly reduced, if *shared nodes* are exploited [15]. That is, if a node v has more than one predecessor. Functions $f : \mathbb{B}^n \to \mathbb{B}^m$ (i.e. functions with more than one output) can be represented more compactly using shared nodes. Further reduction can be achieved if *complement edges* [23] are applied. In particular, this enables the representation of a function as well as its negation by a single node only. To keep the descriptions easier, in the following BDDs without complement edges are used for presentation while they are used in the experiments.

EXAMPLE 2. *Figure 2(a) shows a BDD realizing the function $f = x_1 \oplus x_2$. A BDD representing a function containing shared nodes and including two outputs $f_1 = x_1 \wedge x_2$ and $f_2 = x_1 \vee x_2$ is depicted in Figure 3(a). Edges from a node v to $low(v)$ ($high(v)$) are marked with a small* 0 *(*1*).*

3. SYNTHESIS APPROACH

In this section we describe how to derive a reversible circuit from a given BDD representation. First the general idea (i.e. substituting BDD nodes by a cascade of reversible gates) is introduced and discussed. Afterwards the overall synthesis algorithm is described in detail.

3.1 General Idea

Boolean functions can be efficiently represented by BDDs [15]. Having a BDD $G = (V, E)$, a reversible network can be derived by traversing the decision diagram and substituting each node $v \in V$ with a cascade of reversible gates. The respective cascade of gates depends on the concrete type of the node v. For the general case, Figure 4 shows a substitution with two Toffoli gates (and quantum cost of six). This substitution can be applied to derive a complete Toffoli network from a BDD without shared nodes. Thereby, some inputs have to be set to constants, if the respective $low(v)$ or $high(v)$ edge leads to a terminal node.

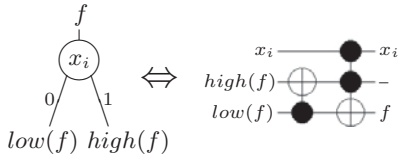
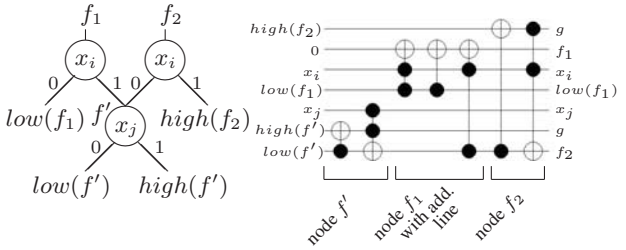


Figure 4: BDD node and equivalent cascade of Toffoli gates



(a) BDD (b) Resulting circuit

Figure 5: Toffoli circuit for shared BDD

EXAMPLE 3. *Consider the BDD in Figure 2(a). Applying the substitution given in Figure 4 to each node of the BDD, the Toffoli network depicted in Figure 2(b) results.*

However, BDD packages make use of shared nodes. But since shared nodes cause fan-outs (which are not allowed in reversible logic) a modified substitution has to be applied. More precisely, the values on the signals representing the shared node in the circuit to be synthesized must be preserved until they are not needed by any of the remaining nodes. The same holds for signals representing select variables of the nodes since they are also often required more than once. For example, in Figure 3(a) the value of variable x_1 is needed by two nodes. As a result, in some cases the values of all inputs of a BDD node have to be preserved. To represent this in reversible logic, i.e. to "emulate" a fan-out, an additional line and an adjusted cascade of gates as depicted in the first row of Table 1 is needed.

EXAMPLE 4. *In Figure 5(a) a partial BDD including a shared node f' is shown. Since the value of node f' is used twice (by nodes f_1 and f_2), an additional line (the second one in Figure 5(b)) and the cascade of gates as depicted in the first row of Table 1 is applied to substitute node f_1. In doing so, the value of f' is still available such that the substitution of node f_2 can be applied. The resulting circuit is given in Figure 5(b).*

Using these two substitutions (the one from Figure 4 and the one from the first row of Table 1), each BDD can be transformed to a Toffoli network. However, as the remaining rows of Table 1 show, better substitutions are possible, if terminal nodes occur as successors. For example, a node v with $low(v) = 0$ (fourth row) can be synthesized with only one Toffoli gate. Identity nodes, i.e. nodes with $low(v) = 0$, $high(v) = 1$, and a select variable x_i (not depicted in Table 1), can be represented by the same line as the input x_i and thus need no additional gate. Furthermore, due to the additional line, which has to be added if either $low(v)$ or $high(v)$ is a terminal, it is also possible to preserve all inputs of a node. In particular for shared nodes, this allows better substitutions.

REMARK 1. *One might expect, that a circuit line can be saved if one of the edges $low(v)$ or $high(v)$ leads to a terminal node. But this is not possible due to reversibility which has to be ensured when synthesizing reversible logic. As an example consider a node v with $high(v) = 0$ (second row of Table 1). Without loss of generality,*

Table 1: BDD nodes and cascade of Toffoli gates with add. lines

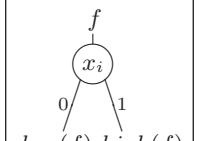
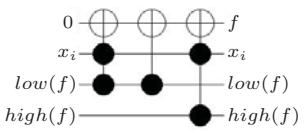
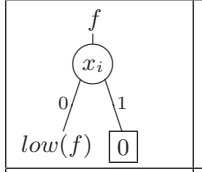
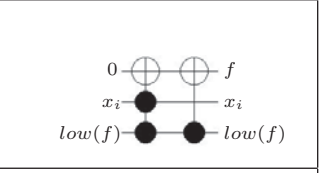
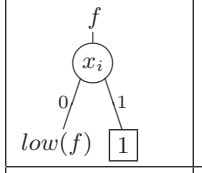
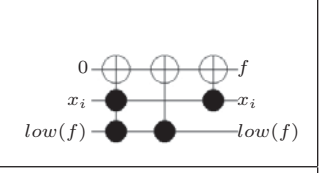
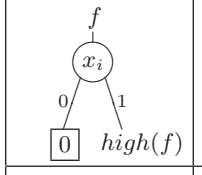
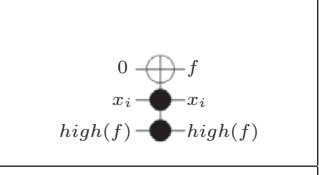
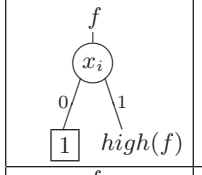
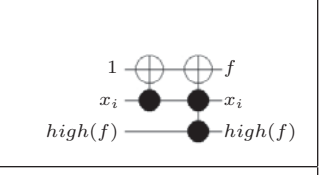
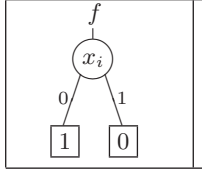
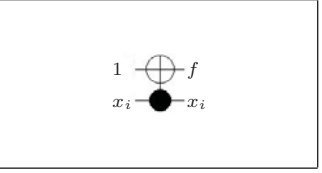
BDD	TOFFOLI CASCADE
	
	
	
	
	
	

Table 2: (Partial) Truth tables for node v with $high(v) = 0$

(a) Without add. line

x_i	$low(x_i)$	f	–
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	?

(b) With additional line

0	x_i	$low(x_i)$	f	x_i	$low(x_i)$
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	1	0	1	1

the first three lines of the corresponding truth table can be embedded with respect to reversibility as depicted in Table 2(a). However, since f is 0 in the last line, no reversible embedding for the whole function is possible. Thus, an additional line is required to make the respective substitution reversible (see Table 2(b))[1].

Similar substitutions can also be applied to synthesize circuits containing elementary quantum gates. This even reduces the resulting quantum cost, since the nodes can be substituted at smaller cost than the Toffoli gate cascades shown above. The same holds, if complement edges are applied that may reduce the size of the BDD. Due to page limitation the concrete substitutions are omitted here but can be found in [24] where these aspects are studied in detail. The respective elementary quantum circuit synthesis as well

[1] Due to the same reason it is also not possible to preserve the values for $low(v)$ or $high(v)$, respectively, in Figure 4.

as the support of complement edges have been considered in the experiments in Section 5.

3.2 Algorithm

Taking the general idea (substituting BDD nodes by cascades of Toffoli or elementary quantum gates, respectively), a method for synthesizing large functions in reversible logic can be formulated: First, a BDD for the function f to be synthesized is created. This can be done efficiently using state-of-the-art BDD packages (e.g. CUDD [25]).

Next, the resulting BDD $G = (V, E)$ is traversed by a depth-first search. For each node $v \in V$, three checks are performed:

1. *Node v represents the identity of a primary input (i.e. select input)*
   In this case no cascade of gates is added to the circuit since the identity can be represented by the same circuit line as the input itself.

2. *Node v contains at least one edge ($low(v)$ or $high(v)$, respectively) leading to a terminal*
   In this case substitutions as depicted in Table 1 are applied, since they often need a smaller number of gates (or quantum cost) and additionally preserve the values of all input signals.

3. *The successors of node v (i.e. $low(v)$ and $high(v)$) are still needed since they represent either shared nodes or the identity of an input variable*
   In this case the substitution depicted in the first row of Table 1 is applied, since this preserves the values of all input signals.

If none of these cases hold, then the general case substitution from Figure 4 is applied.

EXAMPLE 5. *Consider the BDD shown in Figure 3(a) representing the functions $f_1 = x_1 \wedge x_2$ and $f_2 = x_1 \vee x_2$. First, the synthesis approach traverses node f'. But since f' represents the identity of x_2 no gates are added. Instead, the third line of the circuit in Figure 3(b) is used for storing both, the value of the primary input x_2 and the value of f'. Afterwards, for node f_1 the substitution shown in the fourth row of Table 1 is applied. This not only reduces the number of gates (in comparison to the substitution of Figure 4), but also preserves the value of f' which is still needed by node f_2. In doing so, the remaining node f_2 can be substituted which completes the circuit (see Figure 3(b)).*

As a result, circuits are synthesized which realize the given function f. Since, each node of the BDD is only substituted by a cascade of gates, the proposed method has a linear worst case run-time and linear memory complexity with respect to the number of nodes in the BDD. Furthermore, as discussed in the next section, theoretical results on upper bounds known for BDDs can be transferred to reversible circuits using the proposed approach.

4. THEORETICAL ANALYSIS

In previous work, lower and upper bounds for synthesis of *reversible* functions containing n variables have been determined. In [22], it has been shown that there exists a reversible function, that requires at least $(2^n / \ln 3) + o(2^n)$ gates (lower bound). Furthermore, the authors proved that every reversible function can be realized with no more than $n2^n$ gates (upper bound). For a restricted gate library leading to smaller quantum cost and thus only consisting of NOT, CNOT and two-controlled Toffoli gates (the same as applied for the substitutions in Figure 4 and Table 1), functions can be synthesized with at most n NOT gates, n^2 CNOT gates, and $9n2^n + o(n2^n)$ two-controlled Toffoli gates (according to [9]). A tighter upper bound of n NOT gates, $2n^2 + o(n2^n)$ CNOT gates, and $3n2^n + o(n2^n)$ two-controlled Toffoli gates has been proved in [14]. In [26] it has been shown, that linear reversible functions are synthesizeable with CNOT gates only. Moreover, their algorithm never needs more than $\Theta(n^2 / \log n)$ CNOT gates for any linear function f with n variables.

Using the synthesis approach proposed in Section 3, reversible networks for a function f with a size dependent on the number of nodes in the BDD can be constructed. More precisely, let f be a function with n primary inputs which is represented by a BDD containing k nodes. Then, the resulting Toffoli circuit consists of *at most*

- $k+n$ circuit lines (since besides the input lines for each node at most one additional line is added) and
- $3 \cdot k$ gates (since for each node cascades of at most 3 gates are added according to Figure 4 and Table 1, respectively).

Asymptotically, the resulting reversible circuits are bounded by the size of the BDD. Since for BDDs many theoretical results exist (see e.g. [16, 17]), using the proposed synthesis approach, these results can be transferred to reversible logic as well. In the following, we sketch some possible results obtained by this observation.

- A BDD representing a single-output function has 2^n nodes in the worst case. Thus, each function can be realized in reversible logic with at most $3 \cdot 2^n$ gates (thereby at most $2 \cdot 2^n$ CNOTs and $2 \cdot 2^n$ Toffoli gates are needed according to the first row of Table 1).
- A BDD representing a symmetric function has n^2 nodes in the worst case. Thus, each symmetric function can be realized in reversible logic with at most $3 \cdot n^2$ gates (thereby at most $2 \cdot n^2$ CNOTs and $2 \cdot n^2$ Toffoli gates are needed according to the first row of Table 1).
- A BDD representing specific symmetric functions, like `AND`, `OR`, or `EXOR` has a linear size. Thus, there exist a reversible circuit realizing these functions in linear size as well.
- A BDD representing an n-bit adder has linear size. Thus, there exist a reversible circuit realizing addition in linear size as well.

Further results (e.g. tighter upper bounds for general function as well as for respective function classes) are also known (see e.g. [16, 17]). Moreover, in a similar way bounds for elementary quantum circuits can be obtained. However, a detailed analysis of the theoretical results that can be obtained by the BDD-based synthesis is left for future work.

5. EXPERIMENTAL RESULTS

We implemented the proposed synthesis approach in C++ on top of the BDD package CUDD [25]. BDDs are constructed with complement edges and optimized using sifting [27, 24]. Both, Toffoli gate circuits and elementary quantum gate circuits have been synthesized. In this section we document experimental results obtained by our approach and compare them to the results generated by (1) the public available RMRLS approach (described in [13]) using version 0.2 in the default settings and (2) the RMS approach (based on the concepts of [14]) in its most recent version including improved handling of don't care conditions at the output.

As benchmarks we used functions provided in RevLib [28] (including most of the functions which have been previously used to evaluate existing reversible synthesis approaches) as well as from the LGSynth package (a benchmark suite for evaluating irreversible synthesis). Since previous approaches (i.e. RMRLS and RMS) require reversible functions as input, non-reversible functions are embedded into reversible ones (based on the concepts of [22]). For BDD-based synthesis, the original function description has been

Table 3: Experimental results

Function		Previous Approaches							BDD-based Synthesis						
		RMRLS [13]				RMS [14]									
Name	PI/PO	L.	GC	QC	Time	GC	QC	Time	L.	GC	QC	QC_{EQ}	Time	Δ QC (RMRLS)	Δ QC (RMS)
RevLib Functions															
4mod5_8	4/1	5	9	25	0.86	5	9	<0.01	7	8	24	18	<0.01	**-7**	9
decod24_10	2/4	4	11	55	497.51	7	19	<0.01	6	11	27	23	<0.01	**-32**	4
mini-alu_84	4/2	5	21	173	495.61	36	248	<0.01	10	20	60	43	<0.01	**-130**	**-205**
alu_9	5/1	5	9	49	122.48	9	25	0.01	7	9	29	22	**0.01**	**-27**	**-3**
rd53_68	5/3	7	–	–	>500.00	221	2646	0.14	13	34	98	75	**<0.01**	–	**-2571**
hwb5_13	5/5	5	–	–	>500.00	42	214	0.01	28	88	276	205	**0.01**	–	**-9**
sym6_63	6/1	7	36	777	485.47	15	119	0.13	14	29	93	69	**<0.01**	**-708**	**-50**
mod5adder_66	6/6	6	37	529	494.46	35	151	0.06	32	96	292	213	**<0.01**	**-316**	62
hwb6_14	6/6	6	–	–	>500.00	100	740	0.04	46	159	507	375	**<0.01**	–	**-365**
rd73_69	7/3	9	–	–	>500.00	1344	20779	1.93	13	73	217	162	**<0.01**	–	**-20617**
hwb7_15	7/7	7	–	–	>500.00	375	3378	0.18	73	281	909	653	**<0.01**	–	**-2725**
ham7_29	7/7	7	–	–	>500.00	26	90	0.09	21	61	141	107	**<0.01**	–	17
rd84_70	8/4	11	–	–	>500.00	124	8738	9.92	34	104	304	229	**<0.01**	–	**-8509**
hwb8_64	8/8	8	–	–	>500.00	229	3846	0.90	112	449	1461	1047	**0.01**	–	**-2799**
sym9_71	9/1	10	–	–	>500.00	27	201	3.98	27	62	206	153	**<0.01**	–	**-48**
hwb9_65	9/9	9	–	–	>500.00	2021	23311	1.45	170	699	2275	1620	**0.02**	–	**-21691**
cycle10_2_61	12/12	12	26	1435	491.87	41	1837	26.17	39	78	202	164	**0.09**	**-1271**	**-1673**
plus63mod4096_79	12/12	12	–	–	>500.00	24	4873	17.74	23	49	89	79	**0.08**	–	**-4794**
plus127mod8192_78	13/13	13	–	–	>500.00	25	9131	57.16	25	54	98	86	**0.21**	–	**-9045**
plus63mod8192_80	13/13	13	–	–	>500.00	28	9183	57.19	25	53	97	87	**0.20**	–	**-9096**
ham15_30	15/15	15	–	–	>500.00	–	–	>500.00	45	153	309	246	**1.25**	–	–
LGSynth Functions															
xor5	5/1	6	27	387	484.11	8	68	0.01	6	8	8	8	**<0.01**	**-379**	**-60**
9sym	9/1	10	–	–	>500.00	27	201	4.00	27	62	206	153	**<0.01**	–	**-48**
cordic	23/2	∼	∼	∼	∼	∼	∼	∼	52	101	325	247	**0.02**	–	–
bw	5/28	∼	∼	∼	∼	∼	∼	∼	87	307	943	693	**<0.01**	–	–
apex2	39/3	∼	∼	∼	∼	∼	∼	∼	498	1746	5922	4435	**0.24**	–	–
pdc	16/40	∼	∼	∼	∼	∼	∼	∼	619	2080	6500	4781	**0.14**	–	–
seq	41/35	∼	∼	∼	∼	∼	∼	∼	1617	5990	19362	14259	**1.14**	–	–
spla	16/46	∼	∼	∼	∼	∼	∼	∼	489	1709	5925	4372	**0.10**	–	–
ex5p	8/63	∼	∼	∼	∼	∼	∼	∼	206	647	1843	1388	**0.02**	–	–
e64	65/65	∼	∼	∼	∼	∼	∼	∼	195	387	907	713	**0.04**	–	–
cps	24/109	∼	∼	∼	∼	∼	∼	∼	930	2676	8136	6301	**0.10**	–	–
apex5	117/88	∼	∼	∼	∼	∼	∼	∼	1147	3308	11292	8387	**0.14**	–	–
i5	133/66	∼	∼	∼	∼	∼	∼	∼	345	530	1738	1382	**0.09**	–	–
i8	133/81	∼	∼	∼	∼	∼	∼	∼	955	3550	11478	8212	**0.25**	–	–
i6	138/67	∼	∼	∼	∼	∼	∼	∼	280	734	2234	1557	**0.06**	–	–
ex4p	128/28	∼	∼	∼	∼	∼	∼	∼	510	1277	4009	3093	**0.03**	–	–
frg2	143/139	∼	∼	∼	∼	∼	∼	∼	1411	4472	14944	11323	**0.69**	–	–
i4	192/6	∼	∼	∼	∼	∼	∼	∼	729	2115	6827	5158	**0.43**	–	–
i7	199/67	∼	∼	∼	∼	∼	∼	∼	403	941	2953	1996	**0.90**	–	–

used which automatically leads to an embedding. All experiments have been carried out on an AMD Athlon 3500+ with 1 GB of memory. The timeout was set to 500 CPU seconds.

The results are summarized in Table 3. The first columns give the name as well as the number of the primary inputs (PI) and primary outputs (PO) of the original function. In the following columns, the number of lines (L.), the gate count (GC), the quantum cost (QC), and the synthesis time (Time) for the respective approaches (i.e. RMRLS, RMS, and the BDD-based Synthesis) are reported[2]. Note that for the BDD-based synthesis two values for quantum cost are given: QC for the cost of the resulting Toffoli gate circuits and QC_{EQ} if elementary gate circuits are synthesized directly. Furthermore, a '∼' denotes, that an embedding needed by the previous synthesis approaches could not be created within the given timeout. Finally, the last two columns (Δ QC) give the absolute difference of the quantum cost for the resultig circuits obtained by the BDD-based elementary quantum circuit synthesis and the RMRLS or RMS approach, respectively.

As a first result, one can conclude, that for large functions to be synthesized it is not always feasible to create a reversible embedding needed by the previous approaches. Moreover, even if this is feasible, both RMRLS and RMS need a significant amount of run-time to synthesize a circuit from the embedding. As a consequence, for most of the LGSynth benchmarks no result can be generated within the given timeout. In contrast, our BDD approach is able to synthesize circuits for *all* given functions within a few CPU seconds.

Furthermore, although the BDD-based synthesis often leads to larger circuits with respect to gate count and number of lines, the resulting quantum cost are significantly lower in most of the cases (except for *4mod5_8*, *decod24_10*, *mod5adder_66*, and *ham7_29*).

[2]Time for BDD-based Synthesis includes both, the time to build the BDD as well as to derive the circuit from it.

As an example, for *plus63mod4096_79* the BDD-BASED SYNTHESIS synthesizes a circuit with twice the number of lines but with two orders of magnitude fewer quantum cost in comparison to RMS. In the best cases (e.g. *hwb9_65*) a reduction of several thousands in quantum cost is achieved. Note that quantum cost are more important than gate count since they consider gates with more control lines to be more costly. Furthermore, the total number of circuit lines that have been added by the BDD-BASED SYNTHESIS is moderate considering the obtained quantum cost reductions (in particular since all additional lines have constant inputs).

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a synthesis approach which can cope with large functions. The basic idea is to create a Binary Decision Diagram for the function to be synthesized and afterwards substituting each node by a cascade of Toffoli or elementary quantum gates, respectively. Since BDDs may include shared nodes causing fan-outs (which are not allowed in reversible logic), also substitutions including an additional circuit line are proposed.

While previous approaches are only able to handle functions with up to 30 variables at high run-time, our BDD-based approach can synthesize circuits for functions with more than hundred variables in just a few CPU seconds. Furthermore, in most of the cases reductions in the resulting quantum cost have been observed.

In future work, we will focus on the optimization of the resulting circuits. In particular, the number of additional lines should be reduced. Existing approaches (e.g. [12, 29, 30, 31]) provide a good starting point, but mainly focus on reducing quantum cost. Another idea is to adjust the cost function of exact BDD implementations with respect to quantum cost and to synthesize the circuits from the resulting BDDs. Finally, a detailed analysis of the theoretical results that can be obtained by the proposed approach is left for future work.

7. ACKNOWLEDGEMENTS

The authors thank Daniel Große and D. Michael Miller for helpful discussions.

8. REFERENCES

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

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

[3] 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.

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

[5] R. Cuykendall and D. R. Andersen. Reversible optical computing circuits. *Optics Letters*, 12(7):542–544, 1987.

[6] R. C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4:21–40, 1993.

[7] W.N.N. 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*, 25(9):1652–1663, 2006.

[8] 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.

[9] 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.

[10] 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.

[11] P. Kerntopf. A new heuristic algorithm for reversible logic synthesis. In *Design Automation Conf.*, pages 834–837, 2004.

[12] D. Maslov, G. W. Dueck, and D. Michael Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD*, 24(6):807–817, 2005.

[13] 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.

[14] 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*, 12(4), 2007.

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

[16] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. Society for Industrial and Applied Mathematics, 2000.

[17] R. Drechsler and D. Sieling. Binary decision diagrams in theory and practice. *Software Tools for Technology Transfer*, 3:112–136, 2001.

[18] E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[19] A. Peres. Reversible logic and quantum computers. *Phys. Rev. A*, (32):3266–3276, 1985.

[20] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[21] D. Maslov and D. M. Miller. Comparison of the cost metrics through investigation of the relation between optimal ncv and optimal nct 3-qubit reversible circuits. *IET Computers & Digital Techniques*, 1(2):98–104, 2007.

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

[23] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[24] R. Wille and R. Drechsler. Effect of BDD optimization on synthesis of reversible and quantum logic. *Workshop on Reversible Computation*, 2009.

[25] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.

[26] K. Patel, I. Markov, and J. Hayes. Optimal synthesis of linear reversible circuits. *Quantum Information and Computation*, 8(3-4):282–294, 2008.

[27] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

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

[29] J. Zhong and J.C. Muzio. Using crosspoint faults in simplifying toffoli networks. In *IEEE North-East Workshop on Circuits and Systems*, pages 129–132, 2006.

[30] A.K. Prasad, V.V. Shende, I.L. Markov, J.P. Hayes, and K.N. Patel. Data structures and algorithms for simplifying reversible circuits. *J. Emerg. Technol. Comput. Syst.*, 2(4):277–293, 2006.

[31] D. Y. Feinstein, M. A. Thornton, and D. M. Miller. Partially redundant logic detection using symbolic equivalence checking in reversible and irreversible logic circuits. In *Design, Automation and Test in Europe*, pages 1378–1381, 2008.