Quantified Synthesis of Reversible Logic

Robert Wille[1] Hoang M. Le[1] Gerhard W. Dueck[2] Daniel Große[1]

[1]*Group for Computer Architecture (Prof. Dr. Rolf Drechsler)*
University of Bremen, 28359 Bremen, Germany
{rwille,hle,grosse}@informatik.uni-bremen.de

[2]*Faculty of Computer Science*
University of New Brunswick, Fredericton, NB, Canada
gdueck@unb.ca

Abstract

In the last years synthesis of reversible logic functions has emerged as an important research area. Other fields such as low-power design, optical computing and quantum computing benefit directly from achieved improvements. Recently, several approaches for exact synthesis of Toffoli networks have been proposed. They all use Boolean satisfiability to solve the underlying synthesis problem. In this paper a new exact synthesis approach based on Quantified Boolean Formula (QBF) satisfiability – a generalization of Boolean satisfiability – is presented. Besides the application of QBF solvers, we propose Binary Decision Diagrams to solve the quantified problem formulation. This allows to easily support different gate libraries during synthesis. In addition, all minimal networks are found in a single step and the best one with respect to quantum costs can be chosen. Experimental results confirm that the new technique is faster than the best previously known approach and leads to cheaper realizations in terms of quantum costs.

1. Introduction

The growing challenges in classical *Computer-Aided Design* (CAD), e.g. exponential growth of the number of transistors in an integrated circuit or power consumption, motivate the research of alternative techniques. As a promising one reversible logic has emerged. Applications to low-power design, optical computing and quantum computing exist, which overcome the limits of classical systems [4, 15].

Since fanout and feedback are not allowed in reversible logic, the respective networks have to be a cascade of reversible gates. In the past different types of gates have been introduced, e.g. multiple control Toffoli [19], multiple control Fredkin [8], Peres [16], and elementary quantum gates [1]. Furthermore, several approaches for the synthesis of reversible logic, heuristic as well as exact ones, have been proposed in the last few years (e.g. [17, 13, 10]).

In this paper only exact synthesis is considered. In [24] the authors were able to synthesize minimal networks for functions with up to three variables. A synthesis approach considering elementary quantum gates only is described in [11]. Exact Toffoli network synthesis based on *Boolean Satisfiability* (SAT) has been proposed in [9]. The authors (1) encode the synthesis problem as a SAT instance and (2) solve this instance by using a state-of-the-art SAT solver. Recently, in [22] two improvements of this approach have been introduced. However, in both approaches the synthesis problem is encoded with a significant overhead due to restrictions of the underlying proof engine, i.e. a separate constraint has to be built for each truth table line of the function f to be synthesized. Thus, the instances grow exponentially with respect to the number of input variables of the function f.

In this paper, we present an approach for reversible logic synthesis which leads to a polynomial size encoding by taking advantage of *Quantified Boolean Formula* (QBF) satisfiability – a generalization of Boolean satisfiability. We formulate the exact synthesis problem of a reversible function f as a QBF problem by encoding the cascade structure of a reversible network as a functional composition of *universal gates* and enforcing to meet the specification of f by quantification. Then, the quantified Boolean formula is efficiently solved by applying *Binary Decision Diagrams* (BDDs). This leads to three major improvements: (1) the circuits are synthesized faster than the best previously known approach, (2) all minimal networks are found in a single step which allows to choose the best one with respect to the quantum costs, and (3) different gate libraries are easily supported by a simple extension of the problem formulation.

The rest of this paper is organized as follows. Section 2 introduces the basics of reversible logic, QBF satisfiability and BDDs. In Section 3 the main flow as used in [9, 22] is reviewed and the weaknesses of these approaches are briefly discussed. After this, the QBF formulation and its implementations are proposed in Section 4 and Section 5, respectively. Experimental results are given in Section 6 and finally the paper is concluded in Section 7.

2. Preliminaries

2.1. Reversible Logic

A reversible logic gate realizes an n-input n-output function that maps each possible input vector to a unique output vector. In other words this function is a bijection. Any reversible function can be represented by a sequence of reversible gates. Many reversible gates have been studied. In this paper we focus on multiple control Toffoli [19], multiple control Fredkin [8] and Peres [16] gates, which are defined below.

Definition 1 *Let $X := \{x_1, \ldots, x_n\}$ be the set of domain variables. 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. In this paper we distinguish between three different gates types:*

- *A* multiple control Toffoli gate *has a single target line x_j. The gate maps $(x_1, x_2, \ldots, x_j, \ldots, x_n)$ to $(x_1, x_2, \ldots, x_{i_1}x_{i_2}\cdots x_{i_k} \oplus x_j, \ldots, x_n)$.*
- *A* multiple control Fredkin gate *has two target lines x_{j_1} and x_{j_2}. The values of the target lines are interchanged iff the conjunction of all control lines evaluates to 1.*
- *A* Peres gate *has one control line x_i and two target lines x_{j_1} and x_{j_2}. The gate maps $(x_1, x_2, \ldots, x_{j_1}, \ldots, x_{j_2} \ldots, x_n)$ to $(x_1, x_2, \ldots, x_i \oplus x_{j_1}, \ldots, x_i x_{j_1} \oplus x_{j_2}, \ldots, x_n)$.*

Since all quantum circuits are reversible, to realize a non-reversible function (i.e. a n-input m-output function with $n > m$) it must be embedded into a reversible one. Therefore, it is often necessary to add constant inputs and garbage outputs [12]. The garbage outputs are by definition don't cares and can be left unspecified. Functions with garbage outputs are called *incompletely specified functions* in the following.

Quantum costs are often used to measure the cost of reversible gates. Every reversible gate can be transformed into a sequence of elementary quantum gates [1]. Each elementary gate has a quantum cost of one. For example, a Toffoli gate with two controls has a cost of five; a Fredkin gate with one control has a cost of seven; and a Peres gate has a cost of four. The Peres gate is of interest, since the realization with two Toffoli gates would imply a cost of six. All cost calculations are based on [1].

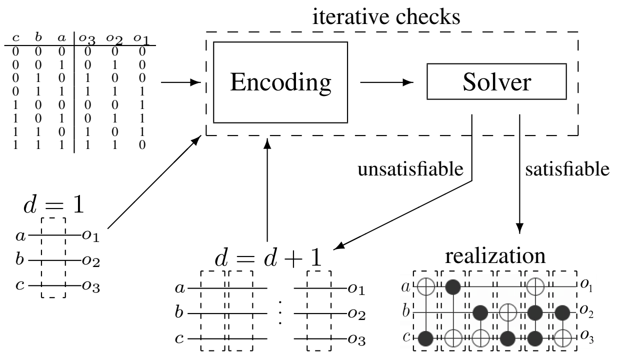
2.2. QBF Satisfiability

The *satisfiability problem* (SAT problem) is to determine whether there exists an assignment of the Boolean variables V for a Boolean function h such that h evaluates to true or to prove that no such assignment exists. That is, SAT asks if $\exists V h$. Thereby, the function is given in *Conjunctive Normal Form* (CNF). A CNF consists of a conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable or its negation. Today, large industrial problems can be solved efficiently by modern *SAT solvers* (see e.g. [14, 7]).

QBF satisfiability is a generalization of SAT, where variables can be universally and existentially quantified. A QBF formula is given in *prenex normal form*, that is $Q_1V_1 \ldots Q_tV_th$, where h is the Boolean function in CNF, $V_i \subset V$ and $Q_i \in \{\exists, \forall\}$. State-of-the-art QBF solvers are e.g. [5, 2].

2.3. Binary Decision Diagram

A Boolean function h can be represented by a *Binary Decision Diagram* (BDD) [6]. A BDD is a directed acyclic graph where a Shannon decomposition $h = \overline{x}_i h_{x_i=0} + x_i h_{x_i=1}$ $(1 \leq i \leq n)$ is carried out in each node. A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain isomorphic sub-graphs nor does it have redundant nodes. Reduced and ordered BDDs are a canonical representation since for each Boolean function the BDD is uniquely specified [6]. In the following, we refer to reduced and ordered BDDs for brevity as BDDs. For construction of BDDs several packages (e.g. CUDD [18]) exists, which provide a wide range of operations.



c	b	a	o_3	o_2	o_1
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	0

Figure 1. Iterative synthesis algorithm

3. Exact Synthesis Procedure

In this section the general procedure of exact synthesis of reversible functions as introduced in [9, 22] is briefly reviewed. The single steps are sketched in Figure 1. As input the reversible function f in terms of a truth table is given. Then, iterative checks are performed. First, the procedure starts with the search for a network realization for f with one gate only ($d = 1$). If it is proven, that no realization with d gates exists, d is incremented. This procedure is repeated until a realization is found. By increasing d iteratively from $d = 1$ minimality is ensured.

For the respective checks, the problem (*Is there a network realization for f with depth d?*) is encoded as an instance of Boolean satisfiability. If an instance becomes satisfiable, a network realization with depth d is found and can be obtained from the satisfying assignment to the variables of the encoding. Otherwise, the instance is unsatisfiable and no realization for the function with depth d exists. For solving the instances in [9] a standard SAT solver has been used while in [22] a specialized proof technique exploiting problem specific knowledge has been applied.

However, in both approaches the problem is encoded for each truth table line separately. That is, the respective constraints representing the network to be synthesized are not built only for one truth table line, but they are duplicated for the remaining $2^n - 1$ truth table lines. Thus, the instances grow exponentially with respect to the number n of variables.

In contrast, using quantified Boolean formulas the problem can be formulated in polynomial size, i.e. the network to be synthesized is encoded only once and the specification of the considered function f is enforced by quantification. We show that this formulation of the synthesis problem in combination with a BDD-based implementation outperforms previous approaches. In addition the proposed approach finds all minimal solutions and thus allows to choose the best one with respect to quantum costs.

4. Problem Formulation Using QBF

In this section the problem formulation for reversible network synthesis using quantified Boolean formulas is introduced. First, the problem formulation for completely specified functions is given. Then, we show how the resulting formulation can be extended for synthesis of incompletely specified functions.

4.1. Completely Specified Functions

In the following the synthesis problem for completely specified functions is defined. For the synthesis of a function f with n inputs/outputs into a reversible network a set $G = \{g_0, \ldots, g_{q-1}\}$ of $q \in \mathbb{N}$ different gates is considered. The set G is used to distinguish between all possible gates in n variables. According to the choosen gate types (i.e. Toffoli, Fredkin and/or Peres) the cardinality of G varies as the following theorem shows:

Theorem 1 *Let f be a reversible function over n variables. Then, there exist*

- $n \cdot 2^{n-1}$ *different multiple control Toffoli gates,*
- $n \cdot (n-1) \cdot 2^{n-2}$ *diff. multiple control Fredkin gates,*
- $n \cdot (n-1) \cdot (n-2)$ *different Peres gates.*

If the gate library used for synthesis consists of more than one gate type, then the numbers above have to be added. For example, in the case of a gate library containing multiple control Toffoli gates and multiple control Fredkin gates for the synthesis of a 3 variable function, G contains $3 \cdot 2^{3-1} + 3 \cdot (3-1) \cdot 2^{3-2} = 12 + 12 = 24$ different gates in total.

Before the synthesis problem is formulated as QBF we define a universal gate that covers the functionality of all gates given in the set G.

Definition 2 *A* universal gate *represents the function*

$$U^G(X,Y) : \mathbb{B}^n \times \mathbb{B}^{\lceil \log q \rceil} \rightarrow \mathbb{B}^n \text{ with}$$

$$U^G(X,Y) = \begin{cases} g_k(X), & \text{if } k = [y_1 \ldots y_{\lceil \log q \rceil}]_2 < q \\ X, & \text{otherwise} \end{cases}$$

where

- $X = \{x_1, \ldots, x_n\}$ *is the set of the inputs of the gate* g_k *and*
- $Y = \{y_1, \ldots, y_{\lceil \log q \rceil}\}$ *is the set of variables representing a binary encoding of a natural number k, which defines the type g_k of the gate (in the following called* gate select inputs*).*

According to the assignments to the gate select inputs Y, a universal gate U^G acts either as a gate from the given set G or as the identity gate[1].

Next, we formalize a reversible network as a cascade of universal gates.

Definition 3 *Let f be a reversible function to be synthesized with at most d gates from the set G. Then, a function F_d is built representing the cascade structure of d universal gates $U^G(X_1, Y_1), \ldots, U^G(X_d, Y_d)$. The output of the i-th universal gate ($0 < i \leq d$) is equal to the input of the next gate, i.e. $U^G(X_i, Y_i) = X_{i+1}$.*

[1]The identity gate has been added to the function definition to handle the case where the set G does not exactly contains a power of two gates. In this case G is extended by identity gates to fill the gap.

Figure 2 shows the resulting cascade structure of the function F_d for d universal gates. Using this structure, any reversible network containing d gates can be obtained by assigning the respective values to each of the gate select input variables $y_{ij} \in Y_i$ ($0 < j \leq \lceil \log q \rceil$).

Based on Definition 3 we have: if a network realization with at most d gates for the reversible function f exists, there has to be at least one assignment to all variables $y_{ij} \in Y_i$ such that F_d is equal to f. More formally, if f is synthesizeable with at most d gates the quantified Boolean formula $\exists y_{11} \ldots \exists y_{d\lceil \log q \rceil} \forall x_1 \ldots \forall x_n (F_d = f)$ holds.

In the next section incompletely specified functions are considered. We show that the above described QBF formulation can be easily adapted for these functions.

4.2. Incompletely Specified Functions

The problem formulation introduced in the last section can be extended to support synthesis of incompletely specified functions as well. Garbage outputs are by definition don't cares and can be left unspecified [12]. The extension of the initial QBF formulation of the synthesis problem is achieved by including the respective *ON-sets* and *don't care sets*.

Definition 4 *Let $f : \mathbb{B}^n \rightarrow \{0, 1, -\}^n$ be an incompletely specified function. Then, f_l^{on} (f_l^{dc}) with $0 < l \leq n$ defines the* ON-set *(*don't care set*) for the l-th output of f.*

We now include in the QBF formulation that the specification is meet if the l-th output of f for a given minterm is don't care. Therefore, we rewrite the QBF formula from above as a conjunction of n one-output functions, where the don't care outputs always evaluate to true, i.e.:

$$\exists y_{11} \ldots \exists y_{d\lceil \log q \rceil} \forall x_1 \ldots \forall x_n (\bigwedge_{l=1}^{n} f_l^{dc} \vee (F_{d_l} = f_l^{on})).$$

The problem considered in the rest of this paper is how to find a satisfying solution for the quantified Boolean formula representing the synthesis problem.

5. Implementations

Based on the QBF formulation for the synthesis problem introduced in the previous section two approaches are used to solve the formula:[2] First, the problem is encoded as an instance of quantified Boolean satisfiability, which is given to a QBF-prover. Second, the function $F_d = f$ is constructed as a BDD and thereafter the quantification is carried out on the BDD. A solution exists if the final BDD is not the constant 0-function. Moreover, all solutions can be extracted by traversing all paths to the 1-terminal.

For both approaches, the incremental nature of F_d is exploited during the construction of the formula. That is, first the formula $F_0 = (x_1, \ldots, x_n)$ is built for depth $d = 0$. Then, for each iteration the function F_d is incrementally built by applying $F_d = U^G(U^G(\ldots(U^G(F_0, Y_1), Y_2)\ldots, Y_{d-1}), Y_d)$. Finally, the equation to f is constrained.

The next two sections describe the steps of both approaches in more detail.

[2]In the following only the implementations for synthesis of completely specified functions are described. Synthesis of incompletely specified functions can be handled analogously.

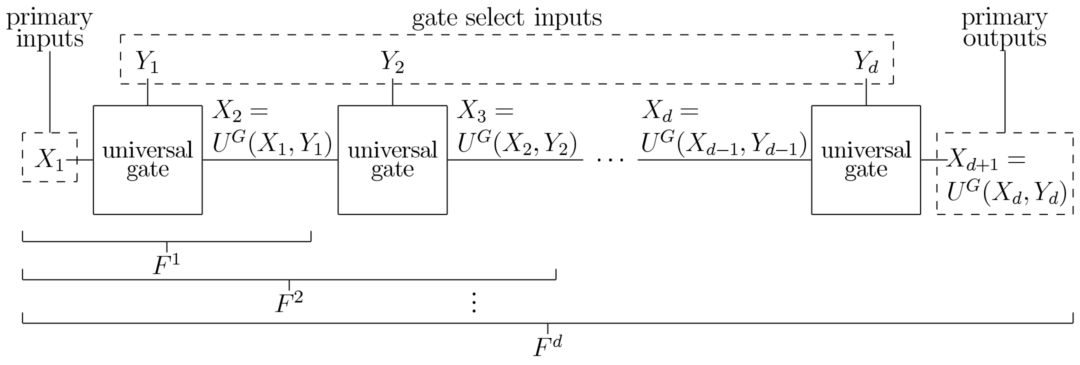


Figure 2. Problem formulation

5.1. Using QBF Solvers

To use a common QBF solver the formula $F_d = f$ has to be transformed into CNF, i.e. a representation has to be created that consists of clauses only. It is known that this can be accomplished in time and space linear in the size of the original Boolean formula [20]. After the transformation of the formula into CNF the resulting set of clauses represents a cascade of d universal gates which has to meet the specification of f. The complete QBF instance is formed by adding the respective existential and universal quantification followed by the existential quantification for the auxiliary variables (denoted as A in the following) added during the transformation into CNF. The resulting overall quantification is: $\exists y_{11} \ldots \exists y_{d\lceil \log q \rceil} \forall x_1 \ldots \forall x_n \exists A$.

Finally, the complete instance is given to a QBF solver. In the case that the instance is satisfiable, a network realization of the circuit can be obtained from the assignments to the variables $y_{ij} \in Y_i$.

5.2. Using BDDs

As shown later in the experiments the performance of the QBF solver approach is low. Therefore, we investigated to perform the synthesis directly with BDDs. That is, instead of building a quantified CNF and solving this instance with a QBF solver, the synthesis is carried out on a BDD representation.

Therefore, we build the BDD for the formula $F_d = f$. This can be done efficiently using a state-of-the-art BDD package. For building the BDD we set the fixed variable order X, Y. The alternative order Y, X leads to a blow up of the BDD representation since in this case the BDD for F_d would already represent all possible functions in n variables which are synthesizable with at most d gates. During the construction isomorphic functions that result from the n output functions for F_d are shared.

After the computation of the equality the resulting BDD is a single output function. For this BDD only the universal quantification of all x_i variables has to be carried out. This is a standard operation available in a BDD package; the idea is to compute the product of the positive co-factor and the negative co-factor for a universally quantified variable, i.e. $\forall x\ h(\ldots, x, \ldots) = h(\ldots, 0, \ldots) \cdot h(\ldots, 1, \ldots)$. If the final BDD consists of the 0-terminal, then no reversible network with the given depth d exists for the function f. Otherwise there is at least one path to the 1-terminal. Each of those paths represents an assignment to all variables $y_{ij} \in Y_i$ and thus, can be converted into a concrete network realization. Since the BDD represents not only one but all 1-paths, in fact *all* realizations with the given depth are found in one single step. All solutions are of interest since one can choose the best mapping to elementary quantum gates which is also shown in the experiments.

6. Experimental Evaluation

The proposed approaches have been implemented in C++. The QBF instances are solved using *sKizzo* [2], a state-of-the-art QBF solver based on symbolic skolemization [3]. For the BDD-based approach the BDD package CUDD [18] version 2.4.1 was used. All approaches have been evaluated on an AMD Athlon 3500+ with 1 GB of main memory. The timeout was set to 2000 CPU seconds. The runtimes of the proposed BDD approach always include the calculation of quantum costs.

6.1. Comparison to Previous Work

In this section we compare quantified synthesis of reversible functions with the SAT-like approaches considered in [22]. In [22], the encoding for the common SAT solver *MiniSat* [7] (denoted by SAT SOLVER) have been compared to a solver utilizing problem specific knowledge (denoted by SWORD according to the name of the solver [21]). In the experiments only synthesis of multiple control Toffoli networks is considered. For comparison we evaluate our approaches with the same set of benchmarks as done in [22] (taken from [23]).[3] The benchmarks include completely specified as well as incompletely specified functions from different problem domains.

The results are given in Table 1. The first column shows the name of the function while D denotes the minimal depth of the resulting network, i.e. the number of Toffoli gates. In the remaining columns the runtimes in CPU seconds (denoted by TIME) and the improvements of the new approaches with respect to the SAT solver (denoted by IMPR$_{SAT}$) and with respect to SWORD (denoted by IMPR$_{SW.}$) are given, respectively. Thereby, the improvement is obtained by the runtime of the SAT/SWORD approach divided by the runtime of the QBF solver/the BDD approach.

From the results it is easy to see that utilizing QBF leads to dramatic improvements for both, the QBF solver and the BDD approach, in comparison to the common SAT

[3]Only some trivial functions (e.g. *peres*, *fredkin*) have been omitted. In contrast we consider two additional functions, i.e. *hwb4* and *4_49* (taken from [23], too). According to the authors of [22] these two functions time out with their approaches.

Table 1. Comparison to Previous Work

		SAT-BASED		QBF-BASED					
		SAT SOLVER	SWORD	QBF SOLVER			BDDS		
BENCH	D	TIME	TIME	TIME	IMPR$_{SAT}$	IMPR$_{SW.}$	TIME	IMPR$_{SAT}$	IMPR$_{SW.}$
COMPLETELY SPECIFIED FUNCTIONS									
mod5mils	5	48.28s	0.08s	32.22s	1.50	<0.01	0.15s	321.87	0.53
graycode6	5	583.14s	0.12s	145.02s	4.02	<0.01	0.46s	1267.69	0.33
3_17	6	0.43s	0.03s	0.19s	2.26	0.16	0.01s	43.00	3.00
mod5d1	7	2094.13s	11.21s	405.96s	5.16	0.03	1.68s	1246.50	6.67
mod5d2	8	1616.17s	9.06s	337.49s	4.79	0.03	3.84s	420.88	2.36
hwb4	11	>2000s	>2000s	>2000s	–	–	20.38s	>98.14	>98.14
4_49	12	>2000s	>2000s	>2000s	–	–	837.92s	>2.39	>2.39
INCOMPLETELY SPECIFIED FUNCTIONS									
rd32-v0	4	2.97s	<0.01s	0.22s	13.50	< 0.05	0.01s	297.00	< 1.00
rd32-v1	5	13.51s	0.04s	0.35s	38.60	0.11	0.03s	450.33	1.33
4mod5-v0	5	122.54s	0.69s	40.01s	3.06	0.02	0.20s	612.70	3.45
4mod5-v1	5	413.21s	0.48s	44.63s	9.25	0.01	0.16s	2582.56	3.00
decod24-v0	6	6.54s	0.02s	0.97s	6.74	0.02	0.04s	163.50	0.50
decod24-v1	6	6.22s	0.09s	1.28s	4.86	0.07	0.04s	155.50	2.25
decod24-v2	6	7.25s	0.02s	1.03s	7.04	0.02	0.03s	241.66	0.66
decod24-v3	7	28.88s	0.18s	2.00s	14.44	0.09	0.05s	577.60	3.60
ALU-v0	6	1998.83s	8.76s	181.99s	10.98	0.05	2.73s	732.17	3.21
ALU-v1	7	>2000s	369.14s	>2000s	–	–	30.42s	>65.75	12.13
ALU-v2	7	>2000s	840.25s	>2000s	–	–	34.72s	>57.60	24.20
ALU-v3	7	>2000s	764.04s	>2000s	–	–	45.69s	>43.77	16.72

Table 2. Quantum costs of networks

BENCH	D	TIME	#SOL	QC
COMPLETELY SPECIFIED FUNCTIONS				
mod5mils	5	0.15	12	13-13
graycode6	5	0.46	1	5-5
3_17	6	0.01	7	14-14
mod5d1	7	1.68	1208	11-15
mod5d2	8	3.84	135	12-20
hwb4	11	20.38	264	23-39
4_49	12	837.92	374	32-72
INCOMPLETELY SPECIFIED FUNCTIONS				
rd32-v0	4	0.01	4	12-12
rd32-v1	5	0.03	20	13-13
4mod5-v0	5	0.20	1176	9-21
4mod5-v1	5	0.16	592	9-25
decod24-v0	6	0.04	75	10-34
decod24-v1	6	0.04	3	14-22
decod24-v2	6	0.03	23	14-26
decod24-v3	7	0.05	1950	11-43
ALU-v0	6	2.73	824	14-38
ALU-v1	7	30.42	850	15-27
ALU-v2	7	34.72	16296	15-55
ALU-v3	7	45.69	132	15-39

solver. Only if additional knowledge is utilized, as done by SWORD, the QBF solver method is outperformed. However, the BDD approach for QBF leads to the smallest overall synthesis time for non-trivial functions. That is, for some benchmarks the runtime is higher than for SWORD indeed, but this only holds for functions with an overall synthesis time of less than one second (e.g. *graycode6* and *decod24-v0*). For all other benchmarks better runtimes are documented. In the best case (*hwb4*) an improvement of a factor of 100 is achieved.

6.2. Quantum Costs of Resulting Networks

After the efficiency of the BDD approach has been shown with respect to the runtime in the last section, further experiments demonstrate the quality of the obtained results.

As described in Section 2.1, after synthesis the resulting Toffoli network has to be transformed into a network consisting of elementary quantum gates only. Thereby, the size of the quantum networks depends on the used Toffoli gates. Thus, it may be an advantage to determine not only one, but more Toffoli network realizations for a given function. Then, by checking the resulting quantum costs for each of the obtained realizations the cheapest one with respect to the quantum costs can be selected.

Previous approaches for minimal Toffoli network synthesis determine only one network in each run. In contrast using BDDs as described in Section 5.2 leads to *all* possible network realizations at once. The differences in the resulting quantum costs are documented in Table 2. Column #SOL denotes the number of solutions found by the BDD approach while QC denotes the minimal as well as the maximal quantum costs for the determined realizations.

Considering the quantum costs of the obtained Toffoli network realization may lead to further significant improvements. For function *4_49* the best realization only needs 32 elementary quantum gates, for example, while in the worst case more than 70 are required. Thus, in contrast to previous algorithms the BDD-based synthesis is not only faster but also another quality criterion – the resulting quantum costs – is applicable.

6.3. Synthesis with Extended Libraries

Finally, in this section we show the application of further gate types to the BDD-based synthesis. This is done by extending the universal gate formula from Section 4 with further gates, i.e. Fredkin and Peres gates.

The results are shown in Table 3. Again, the respective depth (D), the runtime of the synthesis (TIME), the number of solutions (#SOL) and the quantum costs (QC) are listed. Thereby, MCT+MCF denotes the results for a set of gates including multiple control Toffoli and multi-

Table 3. Synthesis Results Using other Gate Libraries

	MCT+MCF				MCT+P				MCT+MCF+P			
BENCH	D	TIME	#SOL	QC	D	TIME	#SOL	QC	D	TIME	#SOL	QC
COMPLETELY SPECIFIED FUNCTIONS												
mod5mils	5	0.52	12	13-13	5	0.90	24	12-13	5	1.50	24	12-13
graycode6	5	2.77	1	5-5	5	4.22	1	5-5	5	7.19	1	5-5
3_17	5	0.02	2	15-15	5	0.02	9	11-12	5	0.03	43	11-20
mod5d1	7	15.08	1352	11-19	7	63.62	5632	10-23	7	250.81	5856	10-25
mod5d2	8	63.96	135	12-20	6	4.61	8	12-19	6	9.60	8	12-19
hwb4	9	37.36	774720	31-51	8	10.03	164	23-29	8	64.83	1084	23-33
4_49	10	1193.90	32	54-58	–	> 2000	–	–	–	> 2000	–	–
INCOMPLETELY SPECIFIED FUNCTIONS												
rd32-v0	4	0.02	4	12-12	2	0.01	4	8-8	2	0.02	4	8-8
rd32-v1	5	0.10	780	11-41	3	0.02	12	9-9	3	0.04	12	9-9
4mod5-v0	5	4.09	3672	9-23	5	2.96	35088	8-27	5	36.32	58176	8-39
4mod5-v1	5	3.52	2792	9-29	4	0.25	768	7-18	4	0.89	768	7-18
decod24-v0	5	0.05	13	11-25	4	0.03	5	13-14	4	0.04	8	13-16
decod24-v1	5	0.04	12	15-23	5	0.05	268	14-27	5	0.09	913	14-29
decod24-v2	6	0.09	1435	12-38	5	0.06	180	11-23	5	0.09	911	11-31
decod24-v3	6	0.07	292	12-32	5	0.06	300	11-29	5	0.09	513	11-31
ALU-v0	4	0.43	22	16-30	6	181.43	29900	12-64	4	1.65	38	16-30
ALU-v1	5	6.73	114	17-33	6	202.87	638	18-32	5	97.95	198	17-33
ALU-v2	5	8.66	224	17-39	6	189.73	280	22-40	5	108.39	402	17-39
ALU-v3	5	10.00	126	17-33	–	> 2000	–	–	5	124.04	431	17-34

ple controle Fredkin gates, MCT+P the results for the set of gates including multiple control Toffoli and Peres gates and MCT+MCF+P the results for the set of all gate types (i.e. multiple control Toffoli, multiple control Fredkin and Peres gates), respectively.

As expected, extending the gate library leads to smaller realizations as for example the result for *hwb4* shows. While the minimal realization of this function only with multiple control Toffoli gates consists of eleven gates it can be reduced by three more gates using additionally Peres gates. Furthermore, improvements with respect to the number of gates can be achieved for *ALU*, *3_17*, *mod5d2*, *4_49*, *rd32* and *decod24*, respectively.

However, with an increasing number of gates to be considered the runtimes increase as well. This can be seen e.g. for function *4_49* or *4mod5*. Only for the functions where the extension of the gate library leads to smaller realizations the runtimes sometimes decrease (e.g. for function *ALU* with the MCT+MCF library) since fewer iterations of the main flow have to be performed (see Section 3).

7. Conclusions

In this paper a new approach for exact synthesis of reversible logic has been proposed. Instead of the representation as an instance of SAT, the problem is formulated as a QBF instance. Since the performance of the QBF solver approach is low, we propose the usage of BDDs. This leads to three major improvements: a runtime speed up of up to a factor of 100, the consideration of another quality criterion (namely the resulting quantum costs), and an easy expandability for further gate libraries using a universal gate formulation. Extensive experimental results clearly demonstrated these improvements.

8. Acknowledgment

We are very thankful to Rolf Drechsler for many helpful discussions and inspirations.

References

[1] 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. *APS Physical Review A*, 52:3457–3467, 1995.

[2] M. Benedetti. sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning, Tech.Rep. 04-11-03, ITC-irst, 2004.

[3] M. Benedetti. sKizzo: a Suite to Evaluate and Certify QBFs. In *Proc. of 20th International Conference on Automated Deduction (CADE05)*, 2005.

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

[5] A. Biere. Resolve and expand. *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, 3542, 2005.

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

[7] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

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

[9] D. Große, X. Chen, G. W. Dueck, and R. Drechsler. Exact SAT-based Toffoli network synthesis. In *Great Lakes Symp. VLSI*, pages 96–101, 2007.

[10] P. Gupta, A. Agrawal, and N. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 25(11):2317–2330, 2006.

[11] W. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. on CAD*, 25(9):1652–1663, 2006.

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

[13] 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.

[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[15] M. A. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

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

[17] V. Shende, A. Prasad, I. Markov, and J. Hayes. Reversible logic circuit synthesis. In *Int'l Conf. on CAD*, pages 353–360, 2002.

[18] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.4.1.* University of Colorado at Boulder, 2005.

[19] 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.

[20] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968.

[21] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler. Sword: A sat like prover using word level information. In *VLSI-SOC*, pages 88–93, 2007.

[22] R. Wille and D. Große. Fast exact Toffoli network synthesis of reversible logic. In *Int'l Conf. on CAD*, pages 60–64, 2007.

[23] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. *RevLib: An Online Resource for Reversible Functions and Reversible Circuits (to be published).* RevLib is avaiable at http://www.revlib.org.

[24] G. Yang, X. Song, W. N. N. Hung, and M. A. Perkowski. Fast synthesis of exact minimal reversible circuits using group theory. In *ASP Design Automation Conf.*, pages 1002–1005, 2005.