Exact SAT-based Toffoli Network Synthesis

Daniel Große
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
grosse@informatik.uni-bremen.de

Xiaobo Chen
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
shoppo@informatik.uni-bremen.de

Gerhard W. Dueck*
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada E3B 5A3
gdueck@unb.ca

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

ABSTRACT

Compact realizations of reversible logic functions are of interest in the design of quantum computers. Such reversible functions are realized as a cascade of Toffoli gates. In this paper, we present the first exact synthesis algorithm for reversible functions using generalized Toffoli gates. Our iterative algorithm formulates the synthesis problem with d Toffoli gates as a sequence of Boolean Satisfiability (SAT) instances. Such an instance is satisfiable iff there exists a network representation with d gates. Thus, we can guarantee minimality. In addition to fully specified reversible functions, the algorithm can be applied to incompletely specified functions. For a set of benchmarks experimental results are given.

Categories and Subject Descriptors

B6.3 [**Design Aids**]: Automatic synthesis

General Terms

Design, Theory

Keywords

Reversible Logic, Quantum Circuits, Synthesis, Minimization, Boolean Satisfiability

1. INTRODUCTION

Reversible logic attracted high attention in the area of low-power design, optical computing and quantum computing. Hence synthesis of reversible logic has become a very important research topic in the last years. In contrast to synthesis with traditional irreversible gates there are two main restrictions for reversible gates: fan-out and feed-back are not allowed. Consequently a network for reversible logic consists of a cascade of reversible gates. The most frequently used gate type is the Toffoli gate [14] which will also be used in this paper. The idea of this gate is to invert one input line (the target line) if the product of a set of control lines evaluates to true.

For the synthesis of reversible logic several approaches have been proposed. In [13] the authors presented an approach based on enumeration and using network equivalences to rewrite a limited set of gates. Other synthesis procedures use heuristics like e.g. spectral techniques [10], positive polarity Reed-Muller expansions [4], or transformation based synthesis [11]. In [9] the authors propose a method that synthesizes the reversible function in a first step and then based on transformations (using so called templates) a realization with fewer gates is computed.

An exact synthesis method based on reachability analysis is described in [5]. However, this procedure is geared towards quantum gates, not Toffoli gates.

In this paper we present the first exact algorithm for Toffoli synthesis of a reversible function[1]. Our method uses an iterative algorithm that is based on *Boolean Satisfiability* (SAT). Hence we can compute a minimal solution in the sense that we can prove that there is no network realization with fewer gates.

The rest of the paper is structured as follows. Section 2 presents the background on reversible logic and Toffoli gates. Furthermore SAT is briefly reviewed. In Section 3 the exact synthesis algorithm is introduced. Experimental results are provided in Section 4. Finally, the paper is summarized.

2. PRELIMINARIES

2.1 Reversible Logic

A reversible logic gate is a n-input n-output function that maps each possible input vector to a unique output vector.

*This work was done while Gerhard W. Dueck was visiting the University of Bremen.

[1]Preliminary results have been presented in [3].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GLSVLSI'07, March 11–13, 2007, Stresa-Lago Maggiore, Italy.
Copyright 2007 ACM 978-1-59593-605-9/07/0003 ...$5.00.

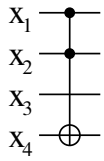

Figure 1: Toffoli gate example

In other words this function is a bijection. Many reversible gates have been studied. Generalized Toffoli gates [14] are widely used. In the rest of this paper we only consider Toffoli gates that are defined as follows:

DEFINITION 1. *Let $X := \{x_1, \ldots, x_n\}$ be the set of domain variables. A* generalized Toffoli gate *has the form $TOF(C,t)$, where $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ is the set of control lines and $t = \{x_j\}$ with $C \cap t = \emptyset$ is the target line. The gate maps $(x_1, \ldots, x_n)$ to $(x_1, \ldots, x_{j-1}, x_j \oplus x_{i_1} \ldots x_{i_k}, x_{j+1}, \ldots, x_n)$.*

An example for the Toffoli gate $TOF(\{x_1, x_2\}, \{x_4\})$ is shown in Figure 1. This gate maps (x_1, x_2, x_3, x_4) to $(x_1, x_2, x_3, x_4 \oplus x_1 x_2)$. The network is drawn in standard notation (see e.g. [12]).

Due to restrictions in quantum mechanics as network topology only a cascade structure can be used. This structure is simply a number of Toffoli gates in a cascade.

DEFINITION 2. *The* reversible cost *(or simply,* cost*) of an implementation of a reversible function f is defined as the number of gates in the network representation that realizes f.*

The goal of this paper is to provide an algorithm for optimal synthesis of a reversible function, i.e. computing the minimal number of gates for a reversible function. The proposed approach is based on Boolean Satisfiability which is described in the following.

2.2 SAT

The *Boolean Satisfiability problem* (SAT) is defined as follows:

DEFINITION 3. *Let h be a Boolean function in* Conjunctive Normal Form *(CNF), i.e. a product-of-sum representation. Then the SAT problem is to determine whether there exists an assignment to the variables of h such that h evaluates to true or to prove that no such assignment exists.*

SAT is one of the central NP-complete problems. In fact, it was the first known NP-complete problem which was proven by Cook in 1971 [1]. Today SAT is not only used in theorem proving, but in many application domains like automatic test pattern generation, logic synthesis, and verification. In the last ten years significant improvements have been made in the area of SAT solvers. Several powerful tools have been developed that make use of Boolean constraint propagation and efficient learning techniques to speed up the proof process (see e.g. [2]).

Here we briefly review the terms that are used in the context of SAT. A literal is a either a variable or its negation. A clause is a disjunction of literals and a CNF consists of a conjunction of clauses.

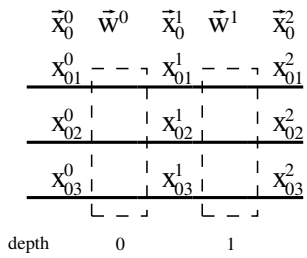

Figure 2: Variables in S_2 with $n = 3, d = 2$ and $i = 0$

EXAMPLE 1. *Let $h = (x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$. Then $x_1 = 1, x_2 = 1$ and $x_3 = 1$ is a satisfying assignment for h. The values of x_1 and x_2 ensure that the first clause becomes 1 while x_3 ensures this for the remaining two clauses.*

3. EXACT SYNTHESIS ALGORITHM

In this section we present the exact synthesis algorithm for reversible logic based on SAT. The basic idea is to check if there exists a Toffoli gate representation for the function with d gates, where d is increased in the next iteration if no realization is found. We first describe a formulation of the synthesis problem with d gates as a Boolean formula. Then, we provide the steps for the transformation of the Boolean formula info a CNF representation. Finally the overall flow of the exact algorithm is presented in detail.

3.1 Boolean Formulation

For a reversible function the synthesis problem is expressed as a Boolean function S_d that is satisfiable iff there exists a Toffoli gate network representation of size d. Before the details for the function S_d are provided the following definitions are given:

DEFINITION 4. *Let $f : B^n \to B^n$ be a reversible function. Then two variable vectors are defined:*

1. *$\vec{x}_i^k = (x_{in}^k x_{i(n-1)}^k \ldots x_{i1}^k)$ with $0 \leq i \leq 2^n - 1$ and $0 \leq k \leq d$ is a Boolean vector representing the input, temporary, or output variables at depth k for line i of the truth-table of f. The left side of the truth-table corresponds to the vector $\vec{x}_i^0$, the right side to the vector $\vec{x}_i^d$, respectively.*
2. *$\vec{w}^k = (w_{\lceil log_2(g) \rceil}^k \ldots w_1^k)$ with $0 \leq k \leq d-1$ is a Boolean vector representing the chosen Toffoli gate at depth k. Here g denotes the number of different Toffoli gates in n variables.*

As an example the variables in the Boolean formulation S_d are shown for the parameters $n = 3, d = 2$ and $i = 0$ in Figure 2. In this example the reversible function has 3 input and 3 output variables, the truth-table has $2^3 = 8$ lines and we are looking for realizations with $d = 2$ Toffoli gates. Remember that the figure only shows the variables for *one* line of the truth-table (here $i = 0$). Furthermore the possible positions for the Toffoli gates are marked with dashed rectangles.

In the following theorem the number of Toffoli gates in n variables is determined.

THEOREM 1. *For a reversible function with n variables there exist $n \cdot 2^{n-1}$ different Toffoli gates.*

PROOF. Since a Toffoli gate has exactly one target line there are $n-1$ lines left as possible control lines. Obviously each possible combination of control lines can be enumerated using the power set of $\{1, \ldots, n-1\}$. In total there are n lines for a reversible function with n variables. So we get $n \cdot 2^{n-1}$ different Toffoli gates. □

Based on the previous definitions and considerations the Boolean formulation S_d for the synthesis of the reversible function $f : B^n \rightarrow B^n$ consists of the conjunction of the following three formulas[2]:

1. The *input-/output constraints* set the input/output pair of each line of the truth-table given by the reversible function f:

$$\bigwedge_{i=0}^{2^n-1} \vec{x}_i^0 = i \wedge \vec{x}_i^d = f(i)$$

2. The *functional constraints* for possible Toffoli gates that are chosen by an assignment to $\vec{w}^k$ are:

$$\bigwedge_{k=0}^{d-1} \bigwedge_{j=0}^{g-1} \bigwedge_{i=0}^{2^n-1} (\vec{w}^k = j) \rightarrow (\vec{x}_i^{k+1} = t(\vec{x}_i^k, j))$$

   These constraints ensure that if at depth k the Toffoli gate j is selected in line i the variables at depth $k+1$ are computed from the variables at depth k with the Toffoli gate function $t(\vec{x}_i^k, j)$. The function $t(\vec{x}_i^k, j)$ is defined by enumerating all possible Toffoli gates with n variables.

3. The *exclusion constraints* ensure that illegal assignments to $\vec{w}^k$ are excluded since not all values of $\vec{w}^k$ are necessary to enumerate all possible Toffoli gates:

$$\bigwedge_{k=0}^{d-1} \vec{w}^k < g$$

Obviously we have found a Boolean formulation for the synthesis problem of the reversible function $f : B^n \rightarrow B^n$ with d generalized Toffoli gates that is satisfiable iff a network realization for f exists with exactly d gates.

3.2 CNF Formulation

For the SAT-based synthesis algorithm we express the Boolean formulation in terms of a CNF. In the following we discuss the construction of the three constraints as a CNF separately:

1. The input/output constraints can be mapped directly by the use of unit clauses, i.e. the the binary encoding for the value of each input/output vector is used to generate the corresponding clauses.

2. It is well known that by the introduction of new variables the CNF form for any Boolean formula can be produced in time and space linear in the size of the original Boolean formula [6]. For the construction algorithm we first define methods for the simple logic functions like AND, OR, etc. that generate the corresponding clauses. Then we extended this scheme for more complex logic like equality of Boolean vectors. Based on a method that enumerates the different Toffoli gates, the functional constraints can be formulated in CNF.

3. The illegal assignments to $\vec{w}^k$ are expressed by explicitly enumerating all values that are not allowed in form of a blocking clause. E.g. if $n = 3$ we have $3 \cdot 2^{3-1} = 3 \cdot 4 = 12$ different Toffoli gates. The vector $\vec{w}^k$ has the length $\lceil log_2(12) \rceil = 4$. However for $\vec{w}^k$ the values $12, 13, 14, 15$ are illegal. E.g. for $\vec{w}^0$ we add the blocking clause $(\overline{w}_4^0 + \overline{w}_3^0 + w_2^0 + w_1^0)$ to exclude the value 12 since this clause evaluates to false for the assignment $w_4^0 = 1, w_3^0 = 1, w_2^0 = 0, w_1^0 = 0$.

[2]For simplicity, natural numbers are identified with their corresponding binary encoding.

```
(1)  exactSynthesis(f : B^n → B^n)
(2)    /* f is given in form of a truth-table */
(3)    found = false;
(4)    d = 1;
(5)    while (found == false) do
(6)      C = constructCNF(f, d);
(7)      r = callSATSolver(C);
(8)      if (r == satisfiable) then
(9)        /* synthesis result with cost d found */
(10)       A = getAssignment();
(11)       ExtractNetworkFromAssignment(A);
(12)       found = true;
(13)     else
(14)       /* no synthesis result with cost d */
(15)       d = d + 1;
(16)     end–if
(17)   end–while
```

Figure 3: Overall flow of exact synthesis algorithm

3.3 SAT-based Algorithm

Based on the CNF formulation for the synthesis problem described in the previous section the overall algorithm is shown in Figure 3. The input for the algorithm is the reversible function f as a truth-table. At first the algorithm starts to find a network representation for f with cost 1, since d is initialized to 1. Then, in the while-loop for the current value of d the CNF C is constructed as described above. This CNF is given to a SAT solver an checked for satisfiability. If there exists a satisfying assignment for C a network representing f has been found. This network – a cascade of Toffoli gates – is extracted from the variables of $\vec{w}^k$ and *found* is set to *true* to abort the main loop. If the CNF C is unsatisfiable d is increased which results in a new search with one more Toffoli gate.

3.4 Incompletely Specified Functions

It is well known that many practical logic functions contain *don't care* conditions. That is, the output for certain input combination is irrelevant. Most traditional minimization algorithms take advantage of this condition. In order to make a function reversible, it is often necessary to add constant inputs and garbage outputs [8]. The garbage outputs are by definition don't cares.

Table 1: Embedding $f = a \cdot b$ in a reversible function.

c	a	b	f	g_1	g_2
0	**0**	**0**	**0**	0	0
0	**0**	**1**	**0**	0	1
0	**1**	**0**	**0**	1	0
0	**1**	**1**	**1**	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	0	1	1

Table 2: Function $f = a \cdot b$ with don't cares.

c	a	b	f	g_1	g_2
0	0	0	0	-	-
0	0	1	0	-	-
0	1	0	0	-	-
0	1	1	1	-	-
1	0	0	-	-	-
1	0	1	-	-	-
1	1	0	-	-	-
1	1	1	-	-	-

EXAMPLE 2. *Consider the function $f = a \cdot b$. To make it reversible, we have to add a constant input c and two garbage outputs g_1 and g_2. A possible assignment is shown in Table 1. This is the optimal assignment, since it can be realized with a single Toffoli gate, namely $TOF(\{a, b\}, \{c\})$. However, not all functions are so easily embedded into a reversible function. It is advantageous to leave the don't care conditions unspecified as shown in Table 2.*

Given a non-reversible function with i input variables and o outputs, that is embedded in a reversible function with c constant inputs and g garbage outputs, then the number of unspecified entries in the truth-table is given by the following formula:

$$(2^n - 2^i) \cdot n + g \cdot 2^i$$

where $n = i + c = g + o$.

For example, the parameters for a full adder (listed as *rd*32 in Table 7 are: $i = 3$, $o = 2$, $c = 1$, and $g = 2$. The number of unspecified entries in the truth table are

$$(2^4 - 2^3) \cdot 4 + 2 \cdot 2^3 = 48.$$

That is, 48 of the 64 entries in the truth table are don't cares. Clearly, the full adder is embedded in many different 4-input reversible function. Choosing the optimal embedding is non-trivial.

Most algorithms that have been proposed for reversible logic synthesis start with a fully specified reversible function. No solution to the problem of finding an optimal or near-optimal embedding of a non-reversible function into a reversible one has been proposed. In our SAT-based procedure the finding of a proper embedding is not necessary — the don't care outputs can be left unspecified.

4. EXPERIMENTAL RESULTS

We have implemented the presented synthesis algorithm in C++. As a SAT solver we use MiniSat v1.14 [2]. All experiments have been carried out on an AMD Athlon 3500+ with 1 GB of main memory.

Table 3: Small example

b	a	b'	a'
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Table 4: Synthesis results for small example

depth	variables	clauses	time	result
1	90	288	0.01	UNSAT
2	172	560	0.01	SAT
3	254	832	0.01	UNSAT
4	336	1,104	0.01	SAT
5	418	1,376	0.01	SAT
6	500	1,648	0.01	SAT

In a first experiment we tested our synthesis algorithm for a very small example. The truth table of the reversible function is shown in Table 3. The function maps (b, a) to $(a \oplus b, b)$. The result of our iterative SAT-based synthesis algorithm[3] is shown in Table 4. The first column gives the depth d for which the CNF is constructed and afterwards the SAT solver is called. The next two columns provide information on the variables and clauses used in the CNF formulation. In column *time* the CPU time for the current iteration in seconds is reported. Then, in the last column it is shown whether the CNF is satisfiable or not. As can be seen for this example several SAT solutions have been found, i.e. there exists Toffoli networks with $2, 4, 5$ and 6 gates. The different solutions are shown in Figure 4. For this simple example it is remarkable that there exists a network realization with 2 gates but no realization with 3 gates. This example already shows that in our SAT-based algorithm it is necessary to make a full search by incrementing the depth starting from 1. Starting with a large d and proving that there is no solution with d gates, we can not conclude that there is no solution with less than d gates.

In a second experiment we applied our synthesis algorithm to a number of benchmarks. First, we studied completely specified functions. The results are shown in Table 5.

The first column provides the name of the reversible func-

[3]For this example we modified the algorithm to compute solutions up to depth 6.

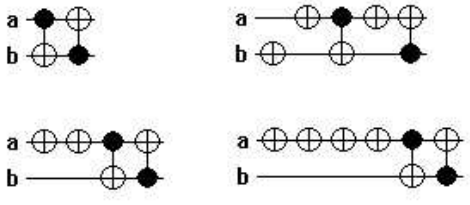

Figure 4: Different network realizations for small example

Table 5: Results for complete specified functions

name	d	vars	clauses	time	res
ham3	1	628	2,524	0.01	U
	2	1,232	5,000	0.02	U
	3	1,836	7,476	0.06	U
	4	2,440	9,952	0.19	U
	5	3,044	12,428	0.32	S
3_17	1	628	2,524	0.01	U
	2	1,232	5,000	0.02	U
	3	1,836	7,476	0.07	U
	4	2,440	9,952	0.12	U
	5	3,044	12,428	0.28	U
	6	3,648	14,904	0.46	S
hwb4	1	3,910	17,632	0.03	U
	2	7,756	35,136	0.12	U
	3	11,602	52,640	0.28	U
	4	15,448	70,144	1.44	U
	5	19,294	87,648	2.74	U
	6	23,140	105,152	7.66	U
	7	26,986	122,656	94.13	U
	8	30,832	140,160	261.38	U
	9	34,678	157,664	2108.82	U
	10	38,524	175,168	22737.50	U
	11	42,370	192,672	21897.30	S
mod5d1	1	22,407	105,808	0.22	U
	2	44,654	211,296	1.93	U
	3	66,901	316,784	16.60	U
	4	89,148	422,272	36.43	U
	5	111,395	527,760	190.79	U
	6	133,642	633,248	1600.46	U
	7	155,889	738,736	215.16	S
graycode6	1	120,968	591,040	2.80	U
	2	241,552	1,181,312	37.10	U
	3	362,136	1,771,584	114.54	U
	4	482,720	2,361,856	107.55	U
	5	603,304	2,952,128	309.00	S

Table 6: Results for incomplete specified functions

name	d	vars	clauses	time	res
decod24-v0	1	3,910	17,584	0.08	U
	2	7,756	35,088	0.22	U
	3	11,602	52,592	0.51	U
	4	15,448	70,096	1.28	U
	5	19,294	87,600	3.40	U
	6	23,140	105,104	2.29	S
decod24-v1	1	3,910	17,584	0.03	U
	2	7,756	35,088	0.11	U
	3	11,602	52,592	0.24	U
	4	15,448	70,096	0.74	U
	5	19,294	87,600	2.04	U
	6	23,140	105,104	3.06	S
decod24-v2	1	3,910	17,584	0.04	U
	2	7,756	35,088	0.11	U
	3	11,602	52,592	0.26	U
	4	15,448	70,096	0.66	U
	5	19,294	87,600	2.18	U
	6	23,140	105,104	4.05	S
decod24-v3	1	3,910	17,584	0.03	U
	2	7,756	35,088	0.10	U
	3	11,602	52,592	0.32	U
	4	15,448	70,096	0.46	U
	5	19,294	87,600	1.91	U
	6	23,140	105,104	17.38	U
	7	26,986	122,608	8.58	S
rd32a	1	3,910	17,584	0.08	U
	2	7,756	35,088	0.22	U
	3	11,602	52,592	0.91	U
	4	15,448	70,096	2.80	S
rd32b	1	3,910	17,584	0.03	U
	2	7,756	35,088	0.11	U
	3	11,602	52,592	0.40	U
	4	15,448	70,096	1.39	U
	5	19,294	87,600	11.56	S
majority3	1	628	2,508	0.01	U
	2	1,232	4,984	0.02	U
	3	1,836	7,460	0.05	S
4mod5a	1	22,407	105,664	0.45	U
	2	44,654	211,152	3.04	U
	3	66,901	316,640	18.73	U
	4	89,148	422,128	87.52	U
	5	111,395	527,616	13.00	S
4mod5b	1	22,407	105,664	0.24	U
	2	44,654	211,152	2.06	U
	3	66,901	316,640	21.99	U
	4	89,148	422,128	105.11	U
	5	111,395	527,616	277.82	S
ALUc	1	22,407	105,680	0.51	U
	2	44,654	211,168	2.68	U
	3	66,901	316,656	12.05	U
	4	89,148	422,144	120.60	U
	5	111,395	527,632	990.10	U
	6	133,642	633,120	844.83	S

tion. In column *d* the current depth is given. Again, in the following columns the information on the SAT instance, i.e. number of clauses and variables, are shown. The needed run time for each iteration in CPU seconds is reported in column *time*. Finally, in the last column it is shown whether the SAT instance is satisfiable or not. We provide a short description for each benchmark presented in the table. *ham3* is a hamming optimal coding function. The specification of *3_17* can be found on [7]. *hwb4* is the hidden weighted bit function of size 4. The *mod5d1* function realizes the Grover's oracle and *graycode6* computes the gray code. As can be seen for many of the benchmarks Toffoli networks of minimal size can be found fast. Since reversible functions have different levels of complexity and due to the heuristic nature of SAT solvers the run times for the benchmarks differ. From the perspective of SAT we can see that we have some hard SAT instances. E.g. for the satisfiable solution of the instance of *hwb4* with depth 11 the run time of the SAT solver was very high. In contrast the result for *graycode6* was computed very fast.

The results for incompletely specified functions are shown in Table 6. Several options for embedding these functions into reversible ones are available. First, one can chose how to set the *constant inputs*. Second, there may be a choice

Table 7: Cost comparison of synthesis results

function	in	c	out	g	old	src.	SAT
rd32	3	1	2	2	4	[9]	4
ham3	3	0	3	0	5	[9]	5
3_17	3	0	3	0	6	[9]	6
hwb4	4	0	4	0	11	[7]	11
mod5d1	4	1	5	0	8	[7]	7
4mod5	4	1	1	4	5	[7]	5
decode42	2	2	4	0	11	[4]	6
graycode6	6	0	6	0	5	[4]	5
alu	5	0	1	4	18	[4]	6

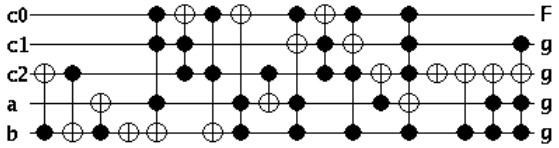

Figure 5: Realization of *alu* from [4]

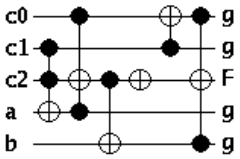

Figure 6: SAT optimized *alu* realization

in where to place the garbage outputs. For example, for the function *decod24* [4] we can chose the values for the constant inputs from $\{00, 01, 10, 11\}$. The results are shown as *decod24-v0* to *decod24-v3* in Table 6. It is apparent, that all four functions exhibit very similar properties, but for one the minimal Toffoli realization requires one more gate. It is therefore important to consider all possible assignments for the constant inputs. For the full adder *rd32* we show the two experiments where the constant was set to zero and one, respectively. As explained it was not necessary to specify values for the garbage outputs since these are handled as don't cares. Due to page limitation we only show the experiment for the best result obtained for *alu* (specification see [4]).

For all benchmark results we give a comparison to the results reported in literature (see e.g. [9]). The comparison is shown in Table 7. The first column provides the name of the function. Columns labeled *in* and *out* give the number of inputs and outputs, respectively. Columns labeled *c* and *g* specify the number or constant inputs and garbage outputs (if the given function is reversible, then there are no garbage outputs). Column *old* gives the number of Toffoli gates in the previously best known result; the source is given in the next column. Finally, we show the size of our SAT-based result in the last column.

We can prove that for some benchmarks minimal Toffoli gate networks have previously been found. For three functions we found a better solution than previously known; two of them are significantly better. The function *mod5d1* (taken from [7]) does not have garbage outputs and the four inputs are also available as outputs. For this function the size of the network was reduced by one compared to the best known realization. For the benchmark functions *decode42* and *alu* we achieved a reduction of 45% and 67%, respectively. This is due to two factors. First, we are able to find the minimal result. Second, there is no need to embed the non-reversible function in a reversible one. The second point seems to offer significant advantages. The two networks for *alu* are shown in Figure 5 and 6 (g denotes the garbage output).

5. CONCLUSIONS

In this paper we presented an exact synthesis algorithm using generalized Toffoli gates for reversible logic functions. Our algorithm uses SAT techniques to find a network realization for the reversible function. We have demonstrated our synthesis algorithm on a set of benchmarks. On the one hand we were able to prove that synthesis results reported in literature are minimal in the number of Toffoli gates. On the other hand we showed that there exist network realizations with fewer gates for some functions. The algorithm takes full advantage of don't care conditions.

6. REFERENCES

[1] S. Cook. The complexity of theorem proving procedures. In *3. ACM Symposium on Theory of Computing*, pages 151–158, 1971.

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

[3] D. Große, X. Chen, and R. Drechsler. Exact toffoli network synthesis of reversible logic using boolean satisfiability. In *IEEE Dallas/CAS Workshop*, pages 51–54, 2006.

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

[5] 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 of Integrated Circuits and Systems*, 25(9):1652–1663, 2006.

[6] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[7] D. Maslov. *Reversible Logic Synthesis Benchmarks Page*. http://www.cs.uvic.ca/~dmaslov/.

[8] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(11):1497–1509, 2003.

[9] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(6):807–817, 2005.

[10] D. M. Miller and G. W. Dueck. Spectral techniques for reversible logic synthesis. In *6th International Symposium on Representations and Methodology of Future Computing Technology*, pages 56–62, 2003.

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

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

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

[14] 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.