Reversible Logic Synthesis with Output Permutation

Robert Wille[1] Daniel Große[1] Gerhard W. Dueck[2] Rolf Drechsler[1]

[1]Institute of Computer Science
University of Bremen
28359 Bremen, Germany

[2]Faculty of Computer Science
University of New Brunswick
Fredericton, Canada

{rwille,grosse,drechsle}@informatik.uni-bremen.de
gdueck@unb.ca

Abstract

Synthesis of reversible logic has become a very important research area. In recent years several algorithms – heuristic as well as exact ones – have been introduced in this area. Typically, they use the specification of a reversible function in terms of a truth table as input. Here, the position of the outputs are fixed. However, in general it is irrelevant, how the respective outputs are ordered. Thus, a synthesis methodology is proposed that determines for a given reversible function an equivalent circuit realization modulo output permutation. More precisely, the result of the synthesis process is a circuit realization whose output functions have been permuted in comparison to the original specification and *the respective permutation vector. We show that this synthesis methodology may lead to significant smaller realizations. We apply* Synthesis with Output Permutation *(SWOP) to both, an exact and a heuristic synthesis algorithm. As our experiments show using the new synthesis paradigm leads to multiple control Toffoli networks that are smaller than the currently best known realizations.*

1. Introduction

According to *Moore's Law* the number of transistors in an integrated circuit doubles every 18 months. Due to this exponential growth, physical boundaries will be reached in the near future. Furthermore, power consumption of circuits becomes a major issue. Quantum computers [12] are an alternative to classical systems. Here, information is stored in so called qubits instead of bits. In comparison to present computers, many problems can be handled more efficiently with the help of quantum computers.

Since all quantum computations are reversible, the synthesis of reversible logic has become an intensely studied topic. In contrast to classical irreversible gates, there are restrictions for reversible gates, e.g. fan-out and feed-back are not allowed. Consequently a network for reversible logic consists of a cascade of reversible gates. In the past different types of reversible gates have been introduced, e.g. (multiple control) Toffoli [18] and Fredkin [2] gates, Peres gates [13], and elementary quantum gates [1].

For the synthesis of reversible logic several approaches – heuristic as well as exact ones – have been proposed. A method based on enumeration that uses network equivalences to rewrite a limited set of gates has been presented in [16]. Proposed heuristics methods are based on spectral techniques [10], positive polarity Reed-Muller expansions [5], or transformation based synthesis [11]. In [9] a method is introduced 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. Techniques of group theory can also be used in the synthesis of reversible logic functions [17]. The authors of [15] introduced a non-search based algorithm running transformations to synthesize reversible functions with CNOT gates. Minimal networks for functions with up to three variables have been synthesized by the approach introduced in [23]. An exact synthesis method based on reachability analysis is described in [6]. In [3, 4, 20] approaches based on *Boolean satisfiability* (SAT) and in [22] a method employing *Quantified Boolean Formula* (QBF) satisfiability are used for exact synthesis.

Usually, the specification of the reversible function to be synthesized is given as a truth table. Thus, each output is set to a fixed position. Since in general the output ordering for a given reversible function f is irrelevant, we propose a synthesis methodology that determines an equivalent circuit realization for f modulo output permutation. That is, the result of the synthesis is a circuit whose outputs have been permuted. Note that no extra gates are invested to achieve the output permutation. In fact, output permutation becomes an integral part of the synthesis process such that the final permutation corresponds to an "update" of reversible function specification. Hence, the synthesis result is a circuit realization *and* the computed output permutation vector.

Based on this idea we introduce first algorithms to apply *Synthesis with Output Permutation* (SWOP). The algorithms focus on synthesis of multiple control Toffoli networks. As the main objective the number of gates is minimized as done by many other researchers (see e.g. [5, 9, 11, 15, 16, 20]). The proposed methodology can also be adapted for other gate libraries as well as other objectives.

The application of output permutations has been recognized before in [11]. It was suggested that for functions with few input variables, all output permutations could be considered. However, neither an analysis of the effect of output permutation nor techniques facing the increasing complexity in case of larger circuits have been considered. This work is an initiative to address this missing domain.

To find the best permutation of outputs for a function,

Table 1. Function specification

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

i.e. the one which leads to the smallest network realization, all $n!$ permutations have to be checked in general (where n is the number of variables of the reversible function). We show how this complexity can be reduced for incompletely specified function (i.e. functions with garbage outputs). Furthermore, we present an exact and a heuristic approach applying synthesis with output permutation. We show that significantly smaller networks (even smaller than the ones known as minimal till today) can be obtained if this new synthesis paradigm is used.

The paper is structured as follows. First, preliminaries are given in Section 2. Section 3 describes the general idea of SWOP while Section 4 gives some theoretical consideration. We introduce an exact and heuristic synthesis algorithm which applies output permutation in Section 5. Results are given and discussed in Section 6. Finally, we conclude the paper and give directions for future work in the last section.

2. Preliminaries

To keep the paper self-contained, this section briefly reviews the basics of reversible logic. For a more detailed insight we refer to the respective publications.

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. Many reversible gates have been studied. Multiple control Toffoli gates [18] (also known as generalized Toffoli gates) 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* multiple control 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 $(x_1, \ldots, x_{j-1}, x_j \oplus x_{i_1} \ldots x_{i_k}, x_{j+1}, \ldots, x_n)$. If no control lines are given (C is empty), then the target line is inverted, i.e. the input vector of the gate is mapped to $(x_1, \ldots, x_{j-1}, x_j \oplus 1, x_{j+1}, \ldots, x_n)$.*

Due to restrictions in quantum mechanics the only possible topology for a network is a cascade of gates.

Definition 2 *The* cost *of a reversible network is defined as the number of its gates.*

Note that, as an additional quality criterion for reversible logic also *Quantum Costs* [1] are used in literature. However, in this work we aim to minimize the number of gates which is done by many other researches as well (see also introduction).

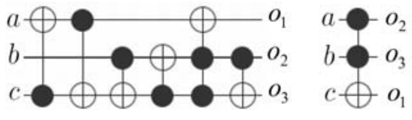


Figure 1. Minimal Toffoli networks

Since all quantum circuits are reversible, to realize a non-reversible function (i.e. an 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 [8]. 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.

3. General Idea

The input of most synthesis approaches is the specification of the reversible function $f : \mathbb{B}^n \to \mathbb{B}^n$ to be synthesized as a truth table. In this table each specified output has a fixed position.

Example 1 *Consider the function specification shown in Table 1. The reversible function maps (c, b, a) to $(b, a, ab \oplus c) = (o_3, o_2, o_1)$. A minimal Toffoli network for this function is shown in Figure 1(a). The cost of this network is* 6.

If the synthesis approach follows the proposed methodology the synthesis result for the given reversible specification is an equivalent circuit realization whose outputs have been permuted.

Example 2 *In Figure 1(b) a Toffoli network is depicted which computes the same reversible function than the Toffoli network shown in Figure 1(a). But in contrast, the three output functions have been "reordered" to another position in the output vector. More precisely, the Toffoli network shown in Figure 1(b) maps (c, b, a) to $(ab \oplus c, b, a) = (o_1, o_3, o_2)$. This reduces the overall costs from* 6 *gates to a single gate, i.e.* 5 *gates have been saved. In total, the result of a synthesis procedure for this example would be the network shown in Figure 1(b)* and *the new output vector (o_1, o_3, o_2).*

Motivated by this example, the question considered in this paper is:

> *How can we efficiently compute good permutations of outputs for a given reversible function to be synthesized such that smaller Toffoli networks result?*

This leads to an extension of common synthesis algorithms we call *Synthesis with Output Permutation* (SWOP) in the rest of this paper. As shown in the following, SWOP may lead to significantly smaller circuits regardless of whether exact or heuristics approaches are used. Since the consideration of all permutations can be expensive with respect to runtime, we propose different strategies which handle the increasing complexity.

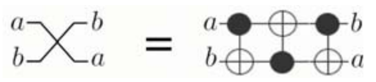


Figure 2. Realization of a permutation

4. Theoretical Consideration

In this section we show the best case benefit that can be achieved by applying SWOP. Therefore, we compare common synthesis to SWOP and determine the maximal number of gates that can be saved if we allow output permutation for an arbitrary reversible function specification. Furthermore, we discuss the worst case complexity to determine the best permutation and show how this can be reduced for incompletely specified functions by exploiting the information on garbage outputs.

4.1. Best Case Benefit

Figure 2 depicts the gates needed to permute two signals in a reversible circuit with multiple control Toffoli gates (in total three gates are required). Since the best position of the outputs is unknown at the beginning of the synthesis process, outputs may be placed arbitrarily in the function specification. Then, the three gates of Figure 2 are needed to permute the value of a signal to the position given by the specification. If in contrast output permutation is considered during the synthesis, the number of gates of the resulting network may be significantly smaller as the following proposition shows.

Proposition 1 *The number of gates in a reversible circuit obtained by common synthesis approaches may be up to* $3 \cdot (n-1)$ *higher than the number of gates in a circuit where synthesis with output permutation is applied (with* n *is the number of variables).*

Proof: Let c be the minimal costs of a circuit obtained by enabling output permutation during synthesis. To move one output line to the position given by the specification three Toffoli gates are required (see Figure 2). At most $n-1$ lines need to be moved. It follows that the cost of the minimal circuit, where no output permutation is allowed, is less than or equal to $c + 3(n-1)$. ■

4.2. Complexity

Finding the best output permutation causes a significant increase in complexity for synthesis. In general, all possible permutations have to be checked, which results in $n!$ different networks in total.

However, it is well known that many practical logic functions contain garbage outputs (see Section 2). The garbage outputs are by definition don't cares and can be left unspecified. Thus, permutations of the garbage outputs need not be considered. This reduces the complexity for SWOP. Instead of $n!$ only $\frac{n!}{g!}$ different permutations are checked (while n is number of variables and g the number of garbage outputs of the reversible function f).

Example 3 *Figure 3 shows all* $n!$ *possible permutations for an incompletely specified function with* $n = 3$ *variables and* $g = 2$ *garbage outputs (denoted by* g_1 *and* g_2*). Since the garbage outputs are left unspecified, the permutations that only swap garbage outputs can be skipped (i.e. the last three permutations of Figure 3). Thus, only* $\frac{3!}{2!} = 3$ *permutations instead of all* $3! = 6$ *permutations are considered.*

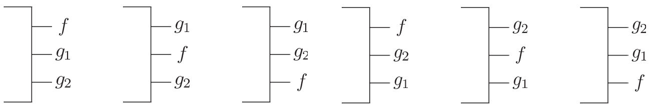


Figure 3. Permutations with garbage outputs

5. Applying Output Permutation

In a naive way, synthesis with output permutation can be easily applied to existing approaches just by encoding all permutations, synthesize each in one turn, and keep the best one. This results in an increase of factor $\frac{n!}{g!}$. In this section we introduce the application of output permutation to exact as well as heuristic approaches using dedicated strategies. Empirical tests show that the increase of the runtime by the proposed approaches is less than the theoretical complexity increase. This is due to the learning technique exploited in the exact approach and due to the heuristic selection of permutations in the heuristic approach.

5.1. Exact Approach

Exact synthesis algorithms determine a *minimal* realization for a given function, i.e. a network with the minimal number of gates. Ensuring minimality is obviously more expensive, but helps e.g. to synthesize smaller networks (or compositions of networks) and to define lower bounds for heuristic approaches. Thus, research in this area is essential.

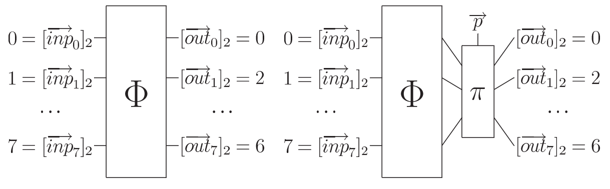
Recently exact algorithms for synthesis of multiple control Toffoli gates using *Boolean satisfiability* (SAT) have been introduced [3, 20]. The basic idea is to check if there exists a Toffoli network representation for a reversible function with c gates (starting with $c = 1$), where c is increased in each iteration if no realization is found. The respective checks are performed by representing the problem as an instance of SAT. This instance is solved by a common SAT solver [3] or by the specialized solve-engine *SWORD*, which additionally uses problem specific knowledge [19, 20]. Due to page limitation we refer to the respective publications for a detailed description of the encodings. In this paper the concrete SAT encoding is simplified as follows:

Definition 3 *Let* $f : \mathbb{B}^n \to \mathbb{B}^n$ *be a reversible function to be synthesized. Then, the SAT instance of the respective synthesis problem is given as*

$$\Phi \wedge \bigwedge_{i=0}^{2^n-1} ([\overrightarrow{inp_i}]_2 = i \wedge [\overrightarrow{out_i}]_2 = f(i)),$$

where

- $\overrightarrow{inp_i}$ *is a Boolean vector representing the inputs of the network to be synthesized for truth table line* i,
- $\overrightarrow{out_i}$ *is a Boolean vector representing the outputs of the network to be synthesized for truth table line* i *and,*
- Φ *is a set of constraints representing the synthesis problem according to [3, 20].*


$0 = [\overrightarrow{inp_0}]_2$
$1 = [\overrightarrow{inp_1}]_2$
$7 = [\overrightarrow{inp_7}]_2$
Φ
$[\overrightarrow{out_0}]_2 = 0$
$[\overrightarrow{out_1}]_2 = 2$
$[\overrightarrow{out_7}]_2 = 6$
$\overrightarrow{p}$
π

(a) original (b) with SWOP extension

Figure 4. Encoding for exact synthesis

As an example Figure 4(a) shows the abstracted representation of the synthesis problem for the function specified in Table 1 (the values of the truth table are given as integers).

To apply SWOP to the exact approach and still ensuring minimality, all permutations are considered. This can be done – as mentioned above – by $\frac{n!}{g!}$ separate synthesis calls. However, exploiting the advanced techniques of the used SAT solvers leads to a faster synthesis. Therefore, just one additional Boolean vector is needed.

Definition 4 *Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ be a reversible function to be synthesized. Then, $\overrightarrow{p} = (p_{\lceil \log_2 \frac{n!}{g!} \rceil}, \ldots, p_1)$ is a Boolean vector representing the binary encoding of a natural number $p \in \{1, \ldots, \frac{n!}{g!}\}$ which indicates the chosen output permutation of the network.*

Using this vector, the SAT encoding is slightly extended: According to the assignments to $\overrightarrow{p}$ (set by the SAT solver) a value for p is determined, which selects the current output permutation. Depending on this permutation the respective output order is set during the search. More formally, the encoding of Definition 3 is extended as follows:

$$\Phi \wedge \bigwedge_{i=0}^{2^n-1} ([\overrightarrow{inp_i}]_2 = i \wedge [\overrightarrow{out_i}]_2 = \pi_{\overrightarrow{p}}(f(i)))$$

The extended encoding of the synthesis problem for the function specified in Table 1 is shown in Figure 4(b).

If the solver finds a satisfying assignment for the SWOP instance, one can obtain the network from the result as described in [3, 20] *and* the best permutation is provided by the assignment to $\overrightarrow{p}$.

Overall, this extension allows exact SWOP with only one synthesis call in contrast to $\frac{n!}{g!}$ separate ones. Furthermore, since the variables of $\overrightarrow{p}$ are an integral part of the search space, the permutations are checked much more efficiently. Because of modern SAT techniques (in particular *conflict analysis* [7]), during the search process reasons for conflicts are learned. This learned information prevents the solver from reentering non-solution search space, i.e. large parts of the search space are pruned. In contrast, this information is not available when each permutation is checked by separate calls of the solver. Thus, exact synthesis with output permutation is possible in feasible runtime when learning is exploited. Experimental results for exact SWOP are given in Section 6.

5.2. Heuristic Approach

To apply SWOP in a heuristic approach, the algorithm presented in [9] is considered. We avoid the construction of all possible permutations which would lead to a complexity increasing of $n!$ since in [9] garbage outputs are not

```
(1)  HeuristicSWOP($f : B^n \rightarrow B^n$)
(2)    /* $f$ is given as truth-table */
(3)    $perm = \{1, 2, \ldots, n\}$;
(4)    $c_{best}$ = synthesize($perm$);
(5)    $best\_perm = perm$;
(6)    for $i = 0$ to $n - 2$ do
(7)      for $j = i + 1$ to $n - 1$ do
(8)        $tmp\_perm$ = swap($perm, i, j$);
(9)        $c_{tmp}$ = synthesize($tmp\_perm$);
(10)       if ($c_{tmp} < c_{best}$)
(11)         $best\_perm = tmp\_perm$;
(12)       end–if
(13)     end–for
(14)     $perm = best\_perm$;
(15)   end–for
```

Figure 5. Heuristic SWOP

supported. We propose a SWOP-based synthesis heuristic using a sifting algorithm inspired by [14] and hence reduce the above complexity to n^2. Because of the heuristic behavior of sifting maybe not the best permutation is determined. However, as the experiments in Section 6 show, significant improvements can be achieved in feasible runtimes.

The pseudo-code for the sifting algorithm is given in Figure 5. First, an initial permutation is chosen and the realization for this specification is synthesized (lines 3 and 4). As initial permutation we used the one given by the specification of the function. The gate count of this first realization is stored. After this, for each output the best position within the current permutation is searched. This is done by swapping the position of the current output with each of the other positions leading to new permutations (line 8). For each of this new permutations the respective realization is synthesized (line 9). If the gate count of such a realization is smaller than the current best known gate count (line 10), the current permutation is stored as being the best one (line 11). When each position for one output have been checked, the best permutation of these checks is used for the remaining outputs (line 14).

In summary, for each of the first $n - 1$ outputs, the algorithm will find a new position, that will result in a realization with the fewest gates – when synthesized with the heuristic algorithm from [9]. Therewith the complexity of SWOP can be reduced while still improving the obtained results as the next section will show.

6. Experimental Results

This section provides experimental results for SWOP. In total four different aspects are studied: (1) the reduction of the complexity of SWOP when garbage outputs are considered, (2) the results of exact SWOP in comparison to previous exact approaches, (3) the results of heuristic SWOP in comparison to the common heuristic approach, and (4) the quality (with respect to the number of gates) of the circuits synthesized by SWOP in comparison to the currently best known realizations.

For exact synthesis we used the algorithm introduced in [20] (the SWOP extension was implemented on the top of this approach). As heuristic approach the template matching algorithm described in [9] has been used. The respective benchmark functions have been taken from [21]. All experiments have been carried out on an AMD Athlon 3500+ with

Table 2. SWOP considering garbage outputs

				SWOP		OPT. SWOP		
BENCH.	n	g	c	$n!$	TIME (S)	$\frac{n!}{g!}$	TIME (S)	IMPR
4mod5	5	4	5	120	233.18	5	7.37	31.6
decod24	4	0	5	24	0.10	24	0.10	1.0
gt4	4	3	3	24	<0.01	4	<0.01	1.0
gt5	4	3	1	24	0.01	4	<0.01	>1.0
low-high	4	3	4	24	3.71	4	0.39	9.51
0-1-2	4	1	4	24	0.03	24	0.02	1.5
maj4_1	5	4	6	120	3500.90	5	2125.62	1.6
maj4_2	5	4	5	120	191.92	5	4.19	45.8
alu	5	4	6	120	2013.72	5	61.24	32.9
mini_alu_1	4	2	5	24	0.28	12	0.19	1.5
mini_alu_2	5	3	7	120	930.60	20	474.42	1.9
mini_alu_3	5	3	5	120	9.60	20	2.07	4.6

1 GB of main memory. All runtimes are given in CPU seconds. The timeout was set to 3600 CPU seconds (denoted by *TO* in the following).

6.1. SWOP with Garbage Outputs

In a first series of experiments we compare the different complexities which may occur when Toffoli networks for functions containing garbage outputs are synthesized. Here – as described in Section 4.2 – instead of $n!$ permutations only $\frac{n!}{g!}$ are considered.

Table 2 shows a comparison of the exact SWOP approach with both numbers of permutations for each incompletely specified function. The first three columns provide the name of the function, the number n of variables and the number g of garbage outputs, respectively. The minimal costs c (i.e. the minimal number of gates) of a Toffoli network representation is given in column c. Then, the runtimes of SWOP with $n!$ and with $\frac{n!}{g!}$ permutations are given (denoted by TIME). Furthermore, the improvement of the optimized SWOP (i.e. the synthesis with only $\frac{n!}{g!}$ permutations) over SWOP with all $n!$ permutation is provided (i.e. runtime of SWOP divided by runtime of OPT. SWOP).

As expected the reduction of permutations leads to better runtimes for all benchmarks. Improvements up to a factor of 45 can be achieved in the best case.

6.2. Exact SWOP

In this section we compare exact SWOP with the previous exact algorithm from [20]. The results are shown in Table 3.

Here again, the first column provides the name of the function, n and g denote the number of variables and the number of garbage outputs, respectively. The next columns give the minimal costs c determined by the two approaches and the corresponding runtimes. The last column shows information relating the complexity, i.e. the runtime overhead when output permutation is considered ($\frac{\text{SWOP-Time}}{\text{Syn-Time}}$) compared to the factor ($\frac{n!}{g!}$) resulting from the complexity analysis.

It can be seen that for many functions SWOP found smaller networks than the ones generated by the previous exact synthesis approach. Thus, removing the restriction for the output ordering leads to smaller networks for many of well known benchmark functions.

As expected the runtime for SWOP is higher in comparison to the runtime of pure exact synthesis. The reason is that the search space is obviously larger due to the number of output permutations that can be chosen. However, the increase is not as high as the number $\frac{n!}{g!}$. This can be seen in the last column of Table 3. For all benchmarks (except *4mod5* and *3_17*) the runtime of SWOP divided by the runtime of the previous synthesis approach is significantly smaller than the worst case complexity ($\frac{n!}{g!}$). As explained this is due to search space pruning, possible when the encoding is extended such that all permutations can be checked at once. Moreover, for some benchmarks (e.g. *maj4_2* or *alu*) the runtime of SWOP is even smaller than for a single exact solution. This reduction is caused by the fact, that smaller networks are found and thus the synthesis terminates earlier.

Table 3. Exact synthesis vs. exact SWOP

			EXACT SYNTHESIS		EXACT SWOP		$\frac{\text{SWOP-TIME}}{\text{SYN-TIME}}$
BENCH.	n	g	c	TIME (S)	c	TIME (S)	VS. $\frac{n!}{g!}$
4mod5	5	4	5	0.9	5	7.4	$8.4 > 5$
decod24	4	0	6	0.1	**5**	0.1	$1.7 < 24$
gt4	4	3	4	<0.1	**3**	<0.1	$1.0 < 4$
gt5	4	3	3	<0.1	**1**	<0.1	$1.0 < 4$
low-high	4	3	5	0.2	**4**	0.4	$2.2 < 4$
0-1-2	4	1	5	<0.1	**4**	<0.1	$0.5 < 24$
maj4_1	5	4	6	438.0	6	2125.6	$4.8 < 5$
maj4_2	5	4	6	13.6	**5**	4.2	$0.3 < 5$
alu	5	4	7	423.3	**6**	61.2	$0.1 < 5$
mini_alu_1	4	2	5	<0.1	5	0.2	$6.3 < 12$
mini_alu_2	5	3	8	2460.0	**7**	474.4	$0.2 < 20$
mini_alu_3	5	3	5	0.2	5	2.1	$12.2 < 20$
3_17	3	0	6	<0.1	**5**	<0.1	$9 > 6$
graycode6	6	0	5	<0.1	5	13.5	$224.7 < 720$
mod5d1	5	0	7	11.8	7	184.1	$15.6 < 120$
mod5d2	5	0	8	9.9	8	1097.6	$109.9 < 120$
mod5mils	5	0	5	0.1	5	1.7	$21.0 < 120$
rand0	4	0	8	15.3	**7**	26.4	$1.7 < 24$
rand1	4	0	8	5.8	**7**	28.3	$4.9 < 24$
rand2	4	0	9	154.5	**8**	150.3	$1.0 < 24$
rand3	4	0	9	231.5	9	1895.6	$8.2 < 24$
rand4	4	0	9	151.1	9	569.9	$3.8 < 24$

6.3. Heuristic SWOP

In this section we compare the results of heuristic synthesis with output permutation. In fact, the results obtained by common heuristic synthesis (according to [9] in its newest version) are compared with SWOP when all permutations are considered (ALL PERMS) and with SWOP when the sifting algorithm introduced in Section 5.2 is used (SIFTING).

The results are given in Table 4 showing the gate counts of the resulting realizations as well as the time needed for their synthesis.

As can be clearly seen, the effect of output permutation is significant for most of the functions. For example, for the function *aj-e13* the realization is reduced by 30 percent from 40 gates to 28 gates. The best absolute reduction of gates can be observed for function *hwb8*. Here, 35 gates are saved in total when output permutation is applied.

But not only the improvements are of interest. Even a comparison of the best and the worst permutation (shown in column c for ALL PERMS) give some interesting insight. For example, consider the function *hwb5*. One output permutation results in a circuit with 38 gates, while another permutation results in 62 gates. Since a heuristic minimization procedure is used, the results will most likely not be optimal. In fact, according to Proposition 1 the difference between the best and the worst permutation for *hwb5* can not be greater than 12 for minimal realizations – yet it is 24.

Table 4. Heur. synthesis vs. heur. SWOP

		HEURISTIC SYNTHESIS		HEURISTIC SWOP			
				ALL PERMS		SIFTING	
BENCH.	n	c	TIME (S)	c	TIME (S)	c	TIME (S)
3_17	3	6	0.03	6-7	0.32	6	0.25
4_49	4	17	0.40	**14**-22	4.09	**16**	1.09
4mod5	5	9	0.03	9-21	10.02	9	0.75
5mod5	6	18	0.13	**14**-37	254.14	18	3.59
aj-e10	5	33	0.63	**22**-51	107.03	**30**	8.21
aj-e11	4	12	0.09	**11**-22	2.46	**11**	0.55
aj-e12	5	26	0.35	**25**-57	103.37	**25**	8.11
aj-e13	5	40	0.97	**28**-51	112.70	**34**	12.31
ex1	3	4	<0.1	4-8	0.08	4	0.06
graycode3	3	2	<0.1	2-5	0.01	2	0.01
graycode4	4	3	0.01	3-9	0.32	3	0.07
graycode5	5	4	0.03	4-13	4.72	4	0.31
graycode6	6	5	0.08	5-18	67.25	5	1.08
hwb3	3	7	0.06	**6**-11	0.32	7	0.29
hwb4	4	15	0.35	**10**-21	3.70	**10**	0.69
hwb5	5	55	1.66	**38**-62	153.71	**44**	16.49
prime5	6	15	0.20	**13**-40	227.05	**13**	3.09
prime5a	6	16	0.10	**14**-41	291.58	**14**	3.92
ham3	3	5	0.01	**3**-5	0.02	**4**	0.03
hwb6	6	125	7.08	–	TO	**91**	89.20
hwb7	7	283	33.26	–	TO	**259**	656.82
hwb8	8	676	152.13	–	TO	**641**	4525.22
ham7	7	23	0.34	–	TO	23	49.47
rd53	7	16	0.26	–	TO	**13**	10.04

Finally it is shown, that sifting provides good results in a fraction of the CPU time. For most functions with more than six variables it is not feasible to minimize the function considering all permutations. However, sifting offers significant improvements for most of these functions (see the bottom rows of Table 4).

6.4. Reductions Achieved by SWOP

Finally, the quality (with respect to the number of gates) of some circuits synthesized by SWOP is compared to the currently best known realizations obtained by common synthesis approaches. Table 5 shows a selection of functions with the gate count of the currently best known realization (BEST KNOWN c). The source of this realization is given in column SRC. The gate count when output permutation is considered is given in column SWOP c.

Synthesis with output permutation enables the realization of smaller networks than the currently best known realizations. As an interesting example the realizations of the *hwb4* function is observed in more detail. For the initial function specification a *minimal* realization with 11 gates have been synthesized by the *exact* approach described in [3]. Now, using output permutation we are able to synthesize a smaller realization with only 10 gates using a *heuristic* approach.

7. Conclusions and Future Work

In this paper we introduced synthesis with output permutation (SWOP). We discussed the best case benefit and introduced different strategies facing the increasing complexity of our new synthesis paradigm. Output permutation have been applied to a representative of exact as well as heuristic synthesis, respectively. On our set of functions we showed that significant reductions (with respect to the number of gates) can be achieved, i.e. considering output permutation is beneficial. For some cases we synthesized multiple control Toffoli networks which are smaller than the currently best known realizations – even smaller than the ones today known as minimal.

Table 5. Best results obtained by SWOP

	BEST KNOWN		SWOP	
BENCH.	c	SCR.	c	Δc
decod24	6	[20]	5	1
alu	7	[20]	6	1
gt5	3	–	1	2
3_17	6	[20]	5	1
4_49	16	[21]	14	2
aj-e13	40	–	28	12
hwb4	11	[3]	10	1

For future work we plan to integrate the proposed methodology in approaches that also consider other cost metrics during synthesis (like e.g. [22, 23]).

8. Acknowledgements

This work was supported by the German Academic Exchange Service (DAAD).

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. *The American Physical Society*, 52:3457–3467, 1995.

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

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

[4] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008.

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

[6] 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.

[7] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

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

[9] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD*, 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. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

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

[14] R.Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Workshop on Logic Synth.*, pages 3a–1–3a–12, 1993.

[15] M. Saeedi, M. Sedighi, and M. S. Zamani. A novel synthesis algorithm for reversible circuits. In *Int'l Conf. on CAD*, pages 65–68, 2007.

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

[17] L. Storme, A. D. Vos, and G. Jacobs. Group theoretical aspects of reversible logic gates. *Journal of Universal Computer Science*, 5:307–321, 1999.

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

[19] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler. Sword: A SAT like prover using word level information. In *VLSI of System-on-Chip*, pages 88–93, 2007.

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

[21] 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 214–219, 2008. RevLib is available at http://www.revlib.org.

[22] R. Wille, H. M. Le, G. W. Dueck, and D. Große. Quantified synthesis of reversible logic. In *Design, Automation and Test in Europe*, pages 1015–1020, 2008.

[23] 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.