Synthesizing Reversible Circuits for Irreversible Functions

D. Michael Miller[1] Robert Wille[2] Gerhard W. Dueck[3]

[1]Department of Computer Science, University of Victoria, Canada
[2]Group for Computer Architecture (*Prof. Rolf Drechsler*), University of Bremen, Germany
[3]Faculty of Computer Science, University of New Brunswick, Canada
mmiller@uvic.ca, rwille@informatik.uni-bremen.de, gdueck@unb.ca

***Abstract*—Many reversible circuit synthesis procedures have been proposed. A common feature of most methods is that the initial specification must be a completely-specified reversible function. However, often the desired functionality is a, possibly incompletely-specified, irreversible function. In this paper, we consider how to fully automate the process of synthesizing a reversible function given an irreversible specification with particular emphasis on how to embed an irreversible function into a reversible specification. Systematic procedures are presented and results for benchmark problems show the methods produce very good results compared to earlier methods.**

I. Introduction

The synthesis of reversible circuits has received much attention [1]–[17] due to the application of these techniques in low-power design and emerging (e.g. quantum, optical, DNA) computing technologies. These synthesis methods generally take a completely-specified reversible function as the starting point. Hence, to realize an irreversible function, which is most often the objective, usually requires that the function is embedded in a larger reversible specification. Such an embedding is not unique and the choice of embedding can have a very significant effect on the quality of the resulting circuit.

The process of embedding an irreversible function into a reversible specification requires the addition of constant inputs and garbage outputs as well as the assignment of *don't-cares* (DC) to ensure reversibility. For small functions, a good embedding can often be found manually based on knowledge of the structure of the irreversible function. In contrast, this becomes harder and often prohibitive as the function size grows.

In this paper, we consider the synthesis of a reversible circuit for an irreversible function to consist of three steps:

1) Embed the target irreversible function in a completely-specified reversible function,
2) Synthesize a circuit for this reversible function, and
3) Perform post-synthesis optimization.

In particular, we consider the embedding issue and the circuit synthesis as separate steps so that any of the previously presented reversible synthesis methods can be applied. However, we also show how knowledge of the nature of the embedding can be used to simplify the synthesis process. Post-synthesis optimization is here restricted to the application of templates as described in [8].

The embedding problem requires the addition of extra lines and the assignment of DC conditions to achieve a completely-specified reversible function. The first is straightforward and we add the minimal number of lines possible [18]. Three methods for automatically assigning the DC are described and compared.

Other approaches to DC assignment when embedding an irreversible function in a reversible specification can be found in [5], [12], [17]. The method in [5] requires a very high number of garbage lines while the method in [12] generates circuits using gates with a high number of controls. The approach in [17] is of note in that it applies to both reversible and certain types of quantum gates. However, it is potentially computationally very expensive as it uses genetic algorithms and results are presented only for small circuits. A SAT-based approach providing exact solutions for small circuits can be found in [19].

Both the embedding and the synthesis approaches employ heuristics and it is not surprising that the results can be very sensitive to the ordering of the function outputs relative to the function inputs. For an n-output function there are $n!$ output permutations and it is thus not feasible to synthesize a circuit for each of them. Therefore, we employ a method based on the 'sifting' method used to find good variable orderings in decision diagrams. This approach examines on the order of n^2 output permutations.

The main contributions of this paper are the study of embedding techniques and the effect of output permutation including the new sifting-based output permutation algorithm. Details of synthesizing a circuit from the reversible specification are not the main focus, and for clarity we consider algorithms employing multi-control Toffoli gates. However, the methods discussed are equally applicable when synthesis methods targeting other reversible gates are used.

After reviewing the necessary background in Section II, we show how irreversible functions can be embedded into reversible specifications in Section III. The DC assignment problem is addressed in Section IV. Then, the synthesis approaches employed in our work are briefly reviewed in Section V. The output permutation via sifting is discussed in Section VI. The paper ends with experimental results in Section VII and conclusions and suggestion for further research in the final section.

II. Background

This section provides the necessary background for the development in this paper. Readers wishing more in-depth treatment should consult the references.

Definition 1:

1) An ***n-variable Boolean function*** $f(X)$, $X = \{x_1, x_2, ..., x_n\}$ is a mapping $f : B^n \rightarrow B$ with $B = \{0, 1\}$.
2) An ***n-input, m-output Boolean function*** is a system of Boolean functions $y_i = f_i(X), 1 \leq i \leq m$, that will be more compactly written as $F(X)$. Such a function is referred to as a ***multiple-output function***. For simplicity, we will term an n-input, m-output function an $n \times m$ function.
3) The multiple-output function $F(X)$ is ***completely-specified*** if the value of every output is defined for every assignment to the inputs, otherwise it is ***incompletely-specified***.
4) $F(X)$ is a ***reversible function*** iff $n = m$, the function is completely-specified and it maps each input assignment to a unique output assignment. A function that is not reversible is termed ***irreversible***.

We denote the truth table for an $n \times m$ function $F(X)$ as $\mathbf{F}$. This table has 2^n rows $\mathbf{F}_0, \mathbf{F}_1, ...,$ corresponding to the assignments to the variables of X ordered in the conventional manner with x_1 as the most significant variable. $\mathbf{F}_i$ will denote the i^{th} row of the table and $\mathbf{F}_{i,j}$ the j^{th} entry in that row, *i.e.* the assignment to f_j. We refer to $\mathbf{F}_0$ as the *first row* in the table, and say that $\mathbf{F}_i$ occurs *earlier* than $\mathbf{F}_j$ if $i < j$.

Property 1: Since a $p \times p$ reversible function F is a permutation mapping 2^p input patterns onto 2^p output patterns, F^{-1} always exists and is the inverse permutation.

A Boolean function $f(X)$ can be expressed as a positive polarity Reed-Muller (PPRM) expansion $f(X) = \bigoplus_{i=0}^{2^n-1} a_i P_i$ where $a_i \in \{0, 1\}$ and P_i is the product term containing those x_j for which the j^{th} position in the binary expansion of i is 1. Since we only deal with the PPRM case we will for brevity refer simply to RM.

The RM spectra of the size n identity function with outputs $y_1, y_2, ...y_n$ has a single nonzero coefficient $a_{2^{i-1}}$ for each y_i with all other coefficients 0.

Definition 2: The ***RM-distance*** of a reversible function from the identity function is the total number of coefficients for which its spectra differ from the spectra of the identity function for the same number of variables.

A reversible function can be realized by a circuit comprised of a cascade of reversible gates with no fan-out or feedback [20]. A reversible gate, itself realizes a reversible function. Many reversible gates have been proposed [20]. We here consider circuits composed of multiple control Toffoli gates which are defined as follows:

Table I
MCT Gate Quantum Cost.

m	$(n-m) \geq$	**cost**	m	$(n-m) \geq$	**cost**
1		1	8	5	62
2		1	8	1	100
3		5	8	0	253
4		13	9	6	74
5	2	26	9	1	128
5	0	29	9	0	509
6	3	38	10	7	86
6	1	52	10	1	152
6	0	61	10	0	1021
7	4	50	> 10	$m-3$	$12m-34$
7	1	80	> 10	1	$24m-88$
7	0	125	> 10	0	2^m-3

Definition 3: A ***multiple control Toffoli gate*** (MCT) with ***target line*** x_j and ***control lines*** $\{x_{i_1}, x_{i_2}, .., x_{i_k}\}$, maps $(x_1, x_2, .., x_j, .., x_n)$ to $(x_1, x_2, .., x_{i_1}x_{i_2}..x_{i_k} \oplus x_j, .., x_n)$. Note that all controls must be 1 for the target to be inverted.

An MCT gate with no controls always inverts the target line and is thus the well-known ***NOT*** gate. An MCT gate with a single control line is called a ***controlled-NOT*** gate. The case of two control lines is the original Toffoli gate.

We follow the normal convention of using a $\bigoplus$ to indicate the target line of an MCT gate and a ● to indicate the control connections. We will use the notation $MCT\{C; t\}$ where C is the set of controls and t is the target.

The number of gates in a circuit is a simple but not very accurate measure of the circuit's complexity. Here, we will consider the *quantum cost* of a circuit which is taken as the sum of the quantum costs of the individual gates with no optimizations of neighboring gates. The quantum cost of an MCT gate is given in Table I (using the calculations introduced in [21] and further optimized in [18] and [22]). In this table, m is the number of control and target lines for the gate and n is the number of circuit lines.

A reversible circuit may have some number of constant inputs and garbage outputs defined as follows:

Definition 4: An ***constant input*** is additional to the primary function inputs and is set to a fixed value for the circuit to achieve the desired functionality.

Definition 5: A ***garbage output*** is additional to the primary function outputs and is thus a don't-care output.

Example 1: Fig. 1 shows a cascade realizing the full adder. This circuit consists of four MCT gates, one constant input, two garbage outputs, and has a quantum cost of 12.

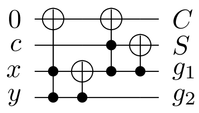


Figure 1. Toffoli gate full adder.

Table II
EMBEDDING THE FULL ADDER INTO A REVERSIBLE FUNCTION.

cxy	CS
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

(a)

acxy	CSgg
0000	00 - -
0001	01 - -
0010	01 - -
0011	10 - -
0100	01 - -
0101	10 - -
0110	10 - -
0111	11 - -
1000	- - - -
1001	- - - -
1010	- - - -
1011	- - - -
1100	- - - -
1101	- - - -
1110	- - - -
1111	- - - -

(b)

acxy	CSgg
0000	0000
0001	0100
0010	0101
0011	1000
0100	0110
0101	1001
0110	1010
0111	1100
1000	0001
1001	0010
1010	0011
1011	0111
1100	1011
1101	1101
1110	1110
1111	1111

(c)

III. EMBEDDING OF IRREVERSIBLE FUNCTIONS

In [18], it was shown that $g = \lceil log_2(\mu) \rceil$ garbage outputs are required to embed a completely-specified irreversible function into a reversible function, where μ is the maximum number of times an output pattern is repeated in the truth table of the irreversible function. For an $n \times m$ irreversible function, this means the reversible function has $m + g$ outputs. Furthermore, c constant inputs must be added such that $n + c = m + g$.

Once the garbage outputs and constant inputs are added, the critical issue is how to assign the DC in the expanded truth table as shown in the following example.

Example 2: Consider the adder function shown in Table II(a). This function has three inputs (the carry-in **c** as well as the two summands **x** and **y**) and two outputs (the carry-out **C** and the sum **S**). This function is clearly irreversible since the number of inputs differs from the number of outputs. Since the output pattern 01 appears 3 times (as does the output pattern 10) adding one additional output to the function (leading to the same number of input and outputs) can not make the function reversible. In fact, $\lceil log_2(3) \rceil = 2$ garbage outputs must be added and hence a constant input must also be added. This is shown in Table II(b). Afterwards, the DC must be assigned so that the resulting function is reversible. One possible, albeit naive, embedding is shown in Table II(c). This embedding was found by assigning the garbage outputs to the patterns 00, 01, and 10 in order for each of the output patterns in the top half of the table and then completing the bottom half of the table using the remaining available output patterns in numerical order.

To appreciate the complexity of choosing a DC assignment, consider Table II(b). There are 4, 4, 3, 4, 2, 3, 2, and 4 choices for completing the DC in the top 8 rows of the table, respectively for a total of 9,216 choices. The bottom 8 rows of the table can then be completed in 8! ways. Lastly, the outputs can be permuted in $4! = 24$ ways. Combining these yields $9216 \times 40320 \times 24 = 8,918,138,880$ possible embeddings for this small example. Since the respective embedding may have a significant effect on the synthesis results (as will be shown in Section IV-D below), finding a 'good' assignment is an important non-trivial task.

IV. DC ASSIGNMENT

In this section, we present three novel methods for DC assignment to complete a reversible specification. We assume that if the irreversible function is incompletely-specified, then the DC in each truth table row will be preassigned to minimize the Hamming distance between the input and the output patterns. We always add the minimal number of garbage outputs and constant inputs to achieve a reversible specification. Lastly, all constant inputs are assigned the value 0 and are always added as the most significant inputs in the truth table. This leads to significant computational advantage as shown below and results in a circuit overhead of at most one NOT gate per constant input.

A. A Greedy Algorithm

The first two methods introduced here for assigning the DC are motivated by the basic operation of transformation based synthesis algorithms (see Section V-A). Those algorithms choose gates to map each row of the truth table to match the corresponding input pattern so that when all rows have been considered the table represents the identity function. It is thus reasonable to conjecture that assigning the DC to minimize the Hamming distance of the output patterns to the corresponding input patterns should help reduce the number of gates required. The first method proposed uses a simple greedy approach.

Algorithm 1: Greedy DC Assignment
Working from the top of the truth table downward:

1) For each distinct output assignment in the irreversible function, identify the target set of rows of the table containing that pattern. Then, determine the set of output assignments which are found by assigning the DC in all possible ways. The candidates are arranged in ascending numerical order.
2) For each row in the target set in turn, choose the first remaining candidate assignment with minimal Hamming distance to the input assignment for that row.

B. Using the Hungarian Algorithm

Let S be the set of truth table rows sharing a common output pattern in the irreversible function and let T be the set of possible assignments to the DC to complete those rows. $|T| = 2^g$ where g is the number of garbage lines added to permit the embedding of the irreversible function into a reversible one. The DC assignment problem is to associate each element of S with a unique element from T. Let $K(S_i, T_j)$ be the 'cost' of associating the DC assignment T_j with S_i. $K(S_i, T_j)$ is the Hamming distance between the completely-specified truth table output pattern and the corresponding input pattern when S_i is completed using T_j. This formulation can be expressed in tabular form with a row for each S_i and a column for each T_j with each $K(S_i, T_j)$ in

Table III
REVERSIBLE EMBEDDINGS FOR THE FULL ADDER.

acxy	Greedy CSgg	XOR CSgg	Naive CSgg	acxy	Greedy CSgg	XOR CSgg	Naive CSgg
0000	0000	0000	0000	1000	1000	1000	0001
0001	0101	0111	0100	1001	0001	1111	0010
0010	0110	0110	0101	1010	0010	1110	0011
0011	1011	1001	1000	1011	0011	0001	0111
0100	0100	0100	0110	1100	1100	1100	1011
0101	1001	1011	1001	1101	1101	0011	1101
0110	1010	1010	1010	1110	1110	0010	1110
0111	1111	1101	1100	1111	0111	0101	1111

the corresponding table entry. Assigning the DC to minimize the total Hamming distance is then a matter of choosing one entry in each row such that those entries appear in unique columns and such that the sum of the chosen entries is minimal. This is a standard assignment problem.

The Hungarian algorithm (also known as the Kuhn-Munkres and Munkres assignment algorithms) is a well-known method [23] for solving the assignment problem in polynomial time. We have implemented this algorithms and results are reported below. The only issue of note here is that storing the potentially very large assignment matrix is avoided since Hamming distance is easily computed as needed – in fact more quickly than a matrix access.

C. An XOR Based Method

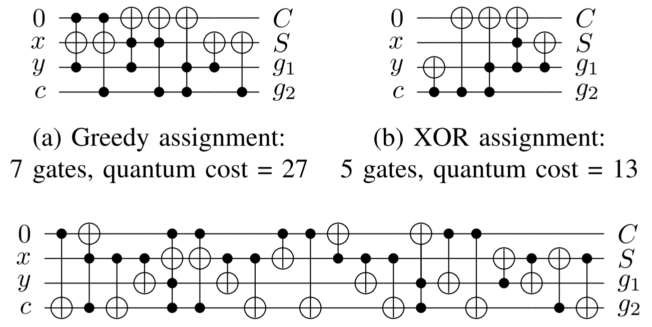
The third method we consider for DC assignment is based on the observation that for many functions a good embedding of an irreversible function into a reversible one is based upon setting the garbage outputs to XOR combinations of the primary inputs. This is often the case for arithmetic functions.

Algorithm 2: XOR-based DC Assignment

```
For $0 \leq i < 2^n$:
  1) Set $k = i$.
  2) Set $p = 0, q = 0$.
  3) For $0 \leq j < n$:
      a) If $f_j$ is a garbage output set $q = q \oplus k_j$, and then set $p_j = q$.
      b) Otherwise set $p_j = \boldsymbol{F}_{ij}$.
  4) If $p$ represents an already assigned output pattern, increment $k$ and repeat from Step 2.
  5) Set $\boldsymbol{F}_i = p$.
```

D. Example

Table III shows embeddings for the full adder using the greedy and the XOR based methods as well as the naive assignment from Table II(c). In this case, the assignment produced by the Hungarian Algorithm method is the same as the assignment for the Greedy method. The circuits for each assignment found using the transformation-based synthesis method [8] (see Section V-A) are shown in Fig. 2. These circuits clearly show the importance of a good DC assignment. The XOR-based method yields a circuit with quantum costs of only 13 while the others yield costs of 27 and 44.



(a) Greedy assignment: 7 gates, quantum cost = 27

(b) XOR assignment: 5 gates, quantum cost = 13

(c) Naive assignment: 20 gates, quantum cost = 44

Figure 2. Circuits for the Embeddings in Table III.

V. SYNTHESIS AFTER DC ASSIGNMENT

After applying one of the above methods, or an alternative, for DC assignment to embed an irreversible function into a completely-specified reversible function, any reversible circuit synthesis method can be applied. We briefly outline two approaches used in the experimental evaluation later in this paper. Furthermore, a combined approach as well as a simplified synthesis method are proposed.

A. Transformation Based Synthesis

Transformation based synthesis for reversible functions was introduced in [7]. The basic idea is to traverse the truth table row by row beginning at the top (the 0^{th}) row. At each step gates are selected to transform the output assignment to match the input assignment. These gates are chosen so that they don't alter rows earlier in the table. After all rows are processed the table represents the identity function and the sequence of gates identified in reverse order is a circuit implementing the given reversible function. The order of gates is reversed because the transformation of output patterns to match input patterns is identifying gates from the output side of the circuit.

Two improvements are possible based on the fact a completely-specified function always has an inverse. First, gates can be chosen to transform an input pattern to match an output pattern if that is cheaper than the other way round. Details can be found in [7]. The second improvement is based on the following result:

Theorem 1: [24] If the cascade of MCT gates $G_1, G_2, ...G_k$ realizes a reversible function F, then the reverse cascade $G_k, ..., G_2, G_1$ realizes the inverse function F^{-1}.

Thus, the transformation based approach can be applied to the inverse of the given function. Since heuristics are used, there can be a significant difference in circuit cost for a function and its inverse. This transformation method was adapted to use the Reed-Muller (RM) spectra of a totally-specified reversible function in [8], [9].

B. Reed-Muller Based Search Algorithm

A second approach introduced in [8] is a very simple search method. At each step, by exhaustive enumeration, it selects the Toffoli gate whose application to the function specification results in the largest decrease of the RM-distance to the identity function (see Definition 2). If no gate application decreases the RM-distance to the identity, a gate is chosen that results in the minimal increase in the RM-distance. In both cases, if there is a tie between two or more gates, the gate with the smallest control set is chosen. If there is a tie based on the number of controls, the method selects the first gate in lexicographic order. The work in [8], [9] showed using RM spectra to be more effective than the Walsh spectrum, used in [25].

This simple search approach has two major drawbacks. First, the algorithm is not guaranteed to converge. Second, the number of possible MCT gates at each stage of the search can be prohibitive.

C. A Combined Algorithm

In [8], the non convergence issue was addressed by first applying the transformation based algorithm and to use the circuit produced as a bound when the search procedure is applied. A more effective alternative is outlined in the following algorithm:

Algorithm 3: Combined Synthesis Algorithm

1) Set $size = 1$.
2) Consider all MCT gates with $size-1$ controls applied to the function specification, *i.e.* at the output side of the circuit.
3) Choose a gate which most reduces the RM-distance to the identity function. If there is a tie the first one encountered is used. Apply the gate. If the function has been mapped to the identity, stop; otherwise go back to Step 1.
4) If no gate can be found that decreases the RM-distance, increment $size$. If $size > maxSize$ go to Step 5, otherwise go to Step 2.
5) Apply the RM spectra-based transformation synthesis algorithm to complete the circuit.

This algorithm incorporates the advantages of the two basic approaches. Using searching first often identifies gates with few controls to begin the circuit that have a strong influence across the function specification. The control value $size$ tends to limit the amount of searching required. We have found that setting $sizeLimit = 5$ yields good results. Resorting to the transformation based approach when searching fails guarantees that a solution is found. The algorithm can be extended by applying bi-directional techniques in both the search and transformation based phases and can be applied to the function and its inverse.

D. Simplified Synthesis

As noted above, we require that all constant inputs take the value 0 and are always the most significant inputs in the truth table. This means that the rows of the irreversible function truth table are always embedded in the first rows of the reversible function truth table while the remaining rows are total DC. In particular, if the reversible function has n primary inputs and c constant inputs we care about the output for the first 2^n rows of the truth table. The remaining $(2^c - 1)2^n$ rows can be ignored.

Clearly, given the above construction, a transformation based synthesis method that works row by row from the top of the truth table can stop after transforming 2^n rows. Because of this, it is not necessary to complete a DC assignment beyond the $(2^n)^{th}$ row.

It is not so obvious that this simplification also applies to both the RM transformation and RM search methods.

1) RM Transformation Synthesis: RM transformation synthesis can stop after the first 2^n rows as described above since, as was shown in [9], if the first p rows of the spectra match the spectra of the identity, then the first p rows in the truth table also match the identity.

2) RM Search Synthesis: Since we only care about the first 2^n rows, we can restrict the gate application to those rows. Furthermore, the RM-distance metric need only be computed over those rows. What goes on below them simply does not matter and has no affect on the first 2^n rows.

Note that when these simplifications are employed, bidirectional synthesis methods can not be applied because we are in fact not working with a completely-specified function and thus have no definition of its inverse. Our experiments show that stopping the synthesis after 2^n rows and avoiding circuitry to align DC rows outweighs the disadvantage of not being able to apply bidirectional techniques.

VI. Optimizations

A. Template Matching

The application of templates to simplify a reversible circuit was introduced in [7] based on the work on circuit transformation rules in [26]. The approach has since been significantly refined [8], [27]. Basically, template matching searches a circuit for a set of gates that either occur together or can be moved to occur together, and then replaces those gates with a less costly set of gates with the same functionality. We here use the templates reported in [8] and use quantum cost as the metric.

Without going into detail, it should be clear that template application can be very computationally expensive due to the amount of searching required including finding appropriate gate repositionings. For that reason, we use a basic set of 14 templates. If the best possible circuit is sought, Dueck and Maslov's [28] 220 templates can be used.

B. Output Permutation

Assuming the correspondence between the inputs and outputs is not fixed a less costly circuit can often be found by reordering the outputs in the specification and resynthesizing. It is not practical to try all $p!$ possibilities for a problem with p outputs. We instead employ sifting as suggested in [29]. A significant new idea here is that rather than using sifting to position the outputs one by one in the order specified, we use the extent to which an output contributes to the RM-distance as a metric to order the sifting of the outputs. The following outlines the basic output sifting approach omitting specific detail on optimization and the avoidance of resynthesizing the same specification.

Algorithm 4: Output Sifting

For a specification with p outputs (including garbage outputs):

1) Assign the DC using one of the methods above. Only the first 2^n rows need be assigned where n is the number of non-constant inputs.
2) Repeat the following p times:
    a) Choose an as yet unsifted output y_k that most contributes to the RM-distance between the specification and the identity function.
    b) Reorder the specification p times placing y_k in each of the p possible positions leaving the relative ordering of the other outputs unchanged. Resynthesize the circuit for each of these repositionings and keep track of the position of y_k that gives the lowest quantum cost circuit.
    c) Put y_k into its best position and record the corresponding circuit as the best seen to date.
3) Report the best circuit found throughout the sifting process.

This algorithm considers p^2 of the $p!$ output orderings so p^2 synthesis attempts are required. Hence the method is expensive to apply for functions with a large number of outputs but it is feasible for p up to 14. Results presented below show it is quite effective.

VII. Experimental Results

Table IV presents results for the incompletely-specified benchmark functions available in RevLib [30] (www.revlib.org). The number at the end of each benchmark name is a unique identifier of the pla-files from RevLib. Results are given for two synthesis methods (combined and transformation based) and for each of the DC assignment methods: greedy, Hungarian algorithm based, and XOR based. The MCT gate count and quantum cost are shown for each circuit. Furthermore, for each benchmark, the best results in terms of quantum cost are highlighted in bold. An AMD Athlon 3500+ with 1 GB of memory was used for our experiments. Every circuit in Table IV was found in less than 0.25 CPU seconds.

The results show no DC assignment method and synthesis method pair is consistently the best. However, since the execution times are relatively short, it is feasible to run all combinations and choose the best circuit obtained. As can be clearly seen, this leads to significant reductions. For example, the synthesis results for function *rd73_69* ranges from quantum cost of 1112 to quantum cost of 184 depending on the DC assignment method applied. An improvement of nearly one order of magnitude is achieved.

Table V shows the effect of the sifting based output permutation method described in Section VI-B followed by template matching from [8]. The benchmark functions from Table IV as well as a number of completely-specified reversible benchmark functions from RevLib are considered. The results labeled 'Initial Synthesis' are for the combined synthesis approach with XOR-based DC assignment used when needed. The section 'Output Permutation (sifting)' shows the results for the same synthesis approach when sifting is employed. The percentage improvement in the circuit quantum cost is shown with a blank entry indicating no improvement.

The section labeled 'Template Matching' shows the result of applying template matching (with 14 templates) to the best circuit found by sifting. Again the improvement shown is with respect to the circuit quantum cost.

The CPU seconds required for the initial synthesis, for output permutation optimization, and for template matching are shown with blank entries indicating the CPU time was less than 0.01 CPU seconds. The time is quite reasonable even for the larger circuits but the results indicate that the time will grow significantly as the number of circuit lines increases. This is largely due to the fact that our current implementation uses truth tables. As anticipated, the CPU requirement for synthesis is roughly $n \times n$ times the time required for the initial synthesis. Template matching is reasonably efficient but the time is significant for the circuits with a large number of gates. However, recall that we here use a set of 14 templates. The time is much higher if a large set of templates is used.

Table V also compares the circuits after template matching with prior results. Those marked with an asterisk are from [8], the others are the best MCT result from the RevLib web site as of March 2009. The quantum cost of our result is shown as a percentage of the cost of the prior result for each function. These comparisons show that the combination of methods described here produce very good results. Indeed there are some cases, where larger circuits are obtained, but recall, that we are comparing the results of our synthesis approach with the best ones obtained by various methods. Furthermore, the circuits quoted from RevLib sometimes uses more lines than the minimal which is what our methods use. If we add these lines and apply some transformations, our results come very close to the RevLib results and in some instances are better.

VIII. Conclusions and Future Research

The first contribution of this paper is the presentation of systematic and complete procedures to determine a reversible circuit for an irreversible function. The second contribution is showing that by assigning the constant inputs to 0 and placing them in the high order variable positions in the truth table, both the DC assignment and the synthesis tasks can be substantially simplified. The third contribution is a sifting method for finding a good output permutation which use the RM-distance metric to determine the order in which the outputs are sifted.

The presented results show that the methods discussed can be quite effective. However, as is often the case in complex

Table IV
COMPARISON OF DC ASSIGNMENT METHODS FOR TWO SYNTHESIS ALGORITHMS

Benchmark				**Combined Synthesis Algorithm**						**Transformation Synthesis Algorithm**					
				Greedy		**Hungarian**		**XOR-based**		**Greedy**		**Hungarian**		**XOR-based**	
	n	**c**	**g**	**gates**	**cost**	**gates**	**cost**	**gates**	**cost**	**gates**	**cost**	**gates**	**cost**	**gates**	**cost**
decod24_10	4	2	0	7	**11**	7	**11**	7	**11**	7	**11**	7	**11**	7	**11**
rd32_19	4	1	2	5	**13**	5	**13**	5	**13**	5	17	5	17	5	17
4gt10_22	5	1	4	3	47	3	47	4	**40**	3	47	3	47	3	47
4gt11_23	5	1	4	1	5	1	**5**	4	12	1	**5**	1	**5**	1	**5**
4gt12_24	5	1	4	3	**55**	3	**55**	6	62	3	**55**	3	**55**	3	**55**
4gt13_25	5	1	4	1	**13**	1	**13**	3	27	1	**13**	1	**13**	1	**13**
4gt4_20	5	1	4	6	**58**	6	**58**	9	65	7	79	7	79	7	79
4gt5_21	5	1	4	3	**19**	3	**19**	6	30	3	**19**	3	**19**	3	**19**
4mod5_8	5	1	4	7	19	7	19	5	9	9	25	9	25	9	25
4mod7_26	5	1	2	21	65	21	65	21	65	15	**55**	15	**55**	15	**55**
alu_9	5	0	4	9	65	16	72	28	224	12	**64**	16	68	32	252
mini-alu_84	5	1	3	22	110	26	114	25	125	22	**98**	28	108	22	110
one-two-three_27	5	2	2	9	**33**	9	**33**	9	**33**	9	**33**	9	**33**	9	**33**
decod24-enable_32	6	3	2	15	39	11	**35**	14	42	15	39	13	37	15	39
rd53_68	7	2	4	27	228	27	228	22	**137**	22	187	22	187	22	187
sym6_63	7	1	6	36	485	36	485	17	**133**	36	777	36	777	36	777
rd73_69	9	2	6	80	1112	80	1112	40	**184**	100	2187	100	2187	100	2187
sym9_71	10	1	9	76	1047	76	1047	51	**573**	210	4368	210	4368	210	4368
rd84_70	11	3	7	104	1823	104	1823	47	**446**	111	2100	111	2100	111	2100

design problems and when heuristics are employed, no single method, or combination of methods, is universally the best. In practice, it is necessary to try different approaches. This can be costly since, while the DC assignment and synthesis methods are themselves relatively fast, the application of output permutation by sifting and template matching can be very costly consuming many minutes of CPU time. We are studying the benchmarks presented here and others to see if there exist properties that will predict which approach would be best for particular types of functions.

Our current implementations are limited as they store the truth tables and the RM spectra as integer vectors. We are in the process of migrating the methods to decision diagram implementations to improve both the efficiency and the size of problem that can be considered. Also, our implementation of sifting for output permutation is very simplistic.

We are investigating how to handle DC in the target irreversible function. One approach [5] is to use a minimizer to assign the DC but that tends to increase the output multiplicity and the number of garbage outputs and constant inputs that must be added. Our current approach is to assign the irreversible function DC to minimize the Hamming distance between the input and output patterns in each truth table row. Future research will consider other alternatives.

REFERENCES

[1] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Reversible logic circuit synthesis," in *Proc. Int'l Conf. on CAD*, 2002, pp. 125–132.

[2] A. Agrawal and N. Jha, "Reversible logic synthesis," in *Proc. Design, Automation and Test in Europe*, 2004, pp. 1384–1385.

[3] J. Donald and N. K. Jha, "Reversible logic synthesis with Fredkin and Peres gates," *J. Emergin Technology Computing Systems*, vol. 4, no. 1, pp. 1–19, 2008.

[4] P. Gupta, A. Agrawal, and N. K. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 25, no. 11, pp. 2317–2330, Nov. 2006.

[5] Z. Guan, X. Qin, Z. Ge, and Y. Zhang, "Reversible synthesis with minimum logic function," in *Proc. Conf. on Computational Intelligence and Security*, vol. 2, 2006, pp. 968–971.

[6] P. Kerntopf, "A new heuristic algorithm for reversible logic synthesis," in *Proc. Design Automation Conf.*, 2004, pp. 834–837.

[7] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proc. Design Automation Conf.*, 2003, pp. 318–323.

[8] D. Maslov, G. W. Dueck, and D. M. Miller, "Techniques for the synthesis of reversible Toffoli networks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 4, pp. 42.1–42.28, 2007.

[9] D. Maslov and D. M. Miller, "Reed-Muller spectra based synthesis of reversible circuits using a quantum cost metric," in *7th International Symposium on Representations and Methodology of Future Computing Technologies (RM)*, 2005.

[10] M. Saeedi, M. Sedighi, and M. S. Zamani, "A novel synthesis algorithm for reversible circuits," in *Proc. Int'l Conf. on CAD*, 2007, pp. 65–68.

[11] M. Saeedi, M. S. Zamani, and M. Sedighi, "On the behavior of substitution-based reversible circuit synthesis algorithms: Investigation and improvement," in *Proc. IEEE Computer Society Symp. on VLSI*, 2007, pp. 428–436.

[12] K. Fazel, M. Thornton, and J. E. Rice, "ESOP-based Toffoli gate cascade generation," in *Proceedings IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2007, pp. 206–209.

[13] Z. Sasanian, M. Saeedi, M. Sedighi, and M. Zamani, "A cycle-based synthesis algorithm for reversible logic," in *Proc. ASP Design Automation Conf.*, 2009, pp. 745–750.

[14] Y. Zheng and C. Huang, "A novel Toffoli network synthesis algorithm for reversible logic," in *Proc. ASP Design Automation Conf.*, 2009, pp. 739–744.

[15] M. Saeedi, M. S. Zamani, and M. Sedighi, "Algebraic characterization of CNOT-baed quantum circuits with its application to logic synthesis," in *Proc. European Conf. on Digital Systems Design*, 2007, pp. 339–346.

[16] D. Yuan, B. Yan, L. WenTao, and G. Yi, "Research and implementation of reversible logic synthesis algorithm in digital system," in *Proc. Computer-Aided Indust. Design and Conceptual Design*, 2006, pp. 1–7.

[17] M. Mohammadi and M. Eshghi, "Heuristic methods to use don't-cares in automated design of reversible and quantum logic circuits," *Quantum Information Processing*, vol. 7, pp. 175–192, 2008.

Table V
EFFECT OF OUTPUT PERMUTATION BY SIFTING FOLLOWED BY TEMPLATE MATCHING

Benchmark		Initial Synthesis			Output Permutation (sifting)				Template Matching				Prior Results		
	n	gates	cost	CPU	gates	cost	%	CPU	gates	cost	%	CPU	gates	cost	%
3_17_6	3	6	14		6	14			6	14			6	14*	100.0%
ex-1_82	3	4	8		4	8			4	8			4	9	88.9%
fredkin_3	3	3	7		3	7			3	7			3	15	46.7%
ham3_28	3	5	9		3	7	22.2%		3	7			5	9*	77.8%
miller_5	3	5	9		5	9			5	9			5	17	52.9%
peres_4	3	2	6		2	6			2	6			2	6	100.0%
toffoli_1	3	1	5		1	5			1	5			1	5	100.0%
4_49_7	4	22	54		16	36	33.3%	0.01	16	36		0.03	12	16	225.0%
aj-e11_81	4	14	34		13	33	2.9%		13	33		0.02	10	30	110.0%
decod24-enable_32	4	14	42		10	34	19.0%		9	33	2.9%	0.01	9	9	366.7%
hwb4_12	4	17	29		12	28	3.4%	0.01	11	23	17.9%	0.02	11	23*	100.0%
rd32_19	4	5	13		5	13			6	10	23.1%		4	12	83.3%
toffoli_double_2	4	2	10		4	8	20.0%		4	8			2	10	80.0%
4gt10_22	5	4	40		4	40			5	37	7.5%		5	37	100.0%
4gt11_23	5	4	12		1	5	58.3%		1	5			4	8	62.5%
4gt12_24	5	6	62		4	36	41.9%		4	36			5	41	87.8%
4gt13_25	5	3	27		1	13	51.9%		1	13			3	15	86.7%
4gt4_20	5	9	65		14	46	29.2%		13	45	2.2%	0.03	6	54	83.3%
4gt5_21	5	6	30		4	16	46.7%		4	16			5	21	76.2%
4mod5_8	5	5	9		5	9			5	9			5	13*	69.2%
4mod7_26	5	21	65		18	62	4.6%		17	57	8.1%	0.06	6	38	150.0%
alu_9	5	28	224		22	110	50.9%		23	107	2.7%	0.08	6	14	764.3%
hwb5_13	5	39	179		48	136	24.0%	0.07	46	126	7.4%	0.26	24	114*	110.5%
mini-alu_84	5	25	125		21	97	22.4%		21	93	4.1%	0.07	6	62	150.0%
mod5d1_16	5	7	15		7	15		0.02	7	11	26.7%	0.01	7	11	100.0%
mod5d2_17	5	8	16		9	13	18.8%	0.02	9	13		0.01	8	16	81.3%
mod5mils_18	5	5	13		5	13		0.02	6	10	23.1%		5	13	76.9%
one-two-three_27	5	9	33		7	27	18.2%		7	27		0.01	8	40	67.5%
decod24_10	6	7	11		6	10	9.1%		6	10		0.01	9	21	47.6%
graycode6_11	6	5	5		5	5		0.05	5	5			5	5	100.0%
hwb6_14	6	169	1129	0.01	101	657	41.8%	0.40	104	556	15.4%	0.88	42	150*	370.7%
mod5adder_66	6	21	121	0.01	21	121		0.32	18	110	9.1%	0.06	17	77	142.9%
ham7_29	7	26	90	0.02	24	72	20.0%	1.06	27	55	23.6%	0.11	25	49*	112.2%
hwb7_15	7	319	3189	0.03	299	2731	14.4%	1.59	299	2450	10.3%	3.40	331	2609*	93.9%
mod10_86	7	33	110		27	88	20.0%	0.03	27	84	4.5%	0.12	7	43	195.3%
rd53_68	7	22	137		23	64	53.3%	0.03	22	59	7.8%	0.10	16	67*	88.1%
sym6_63	7	17	133		16	107	19.5%	0.04	18	93	13.1%	0.06	15	62	150.0%
hwb8_64	8	705	8489	0.10	375	5121	39.7%	6.41	385	4516	11.8%	5.46	748	6197*	72.9%
urf2_73	8	894	10117	0.19	906	9283	8.2%	11.73	900	8843	4.7%	18.11	620	16152	54.7%
hwb9_65	9	2023	23740	0.34	1709	21996	7.3%	24.76	1677	20933	4.8%	57.94	1959	20378*	102.7%
rd73_69	9	40	184	0.01	31	112	39.1%	0.56	29	90	19.6%	0.15	20	76	118.4%
urf1_72	9	2019	24674	0.77	1992	23471	4.9%	62.44	1934	22703	3.3%	76.97	1487	45855	49.5%
urf5_76	9	929	19811	0.39	893	16082	18.8%	37.12	913	14297	11.1%	20.24	499	24253	58.9%
sym9_71	10	51	573	0.03	38	324	43.5%	2.95	39	229	29.3%	0.24	21	94	243.6%
urf3_75	10	3999	59719	2.38	4053	59325	0.7%	255.51	3937	57365	3.3%	340.37	2674	121716	47.1%
rd84_70	11	47	446	0.05	42	441	1.1%	5.86	50	280	36.5%	0.47	15	112	250.0%
cycle10_2_61	12	44	3423	4.86	34	1570	54.1%	563.93	29	1212	22.8%	0.18	19	1206*	100.5%
plus63mod4096_79	12	26	4875	3.26	26	4875		441.04	26	4875		0.12	429	32539	15.0%
plus127mod8192_78	13	27	9133	10.40	27	9133		1673.83	27	9133		0.12	910	73357	12.5%
plus63mod8192_80	13	30	9185	10.42	30	9185		1664.73	30	9185		0.13	492	45025	20.4%

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

[19] R. Wille and D. Große, "Fast exact Toffoli network synthesis of reversible logic," in *Proc. Int'l Conf. on CAD*, 2007, pp. 60–64.

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

[21] A. Barenco, C. Bennett, R. Cleve, D. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, pp. 3457–3467, 1995.

[22] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck, "Quantum circuit simplification using templates," in *Proc. Design, Automation and Test in Europe*, 2005, pp. 1208–1213.

[23] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*. McGraw-Hill Professional, 2005.

[24] D. Maslov, "Reversible logic synthesis," Ph.D. dissertation, University of New Brunswick, 2003.

[25] D. M. Miller and G. W. Dueck, "Spectral techniques for reversible logic synthesis," in *6th Int. Symp. on Representations and Methodology of Future Computing Technologies*, 2003, pp. 56–62.

[26] K. Iwama, Y. Kambayashi, and S. Yamashita, "Transformation rules for designing CNOT-based quantum circuits," in *Proc. Design Automation Conf.*, 2002, pp. 419–424.

[27] D. Maslov, G. W. Dueck, and D. M. Miller, "Fredkin/Toffoli templates for reversible logic synthesis," in *Proc. Int'l Conf. on CAD*, 2003, pp. 256–261.

[28] G. W. Dueck and D. Maslov, "Generation of multiple-control Toffoli network templates," in *Proc. Int'l Workshop on Logic Synthesis*, 2007, pp. 39–44.

[29] R. Wille, D. Große, G. W. Dueck, and R. Drechsler, "Reversible logic synthesis with output permutation," in *Proc. Int'l Conf. on VLSI Design*, 2009, pp. 189–194.

[30] 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*, 2008, pp. 220–225, RevLib is available at www.revlib.org.