A Forward-Looking Non-Search Based Synthesis Algorithm for Reversible Circuits

Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi

Quantum Design Automation Group,
Amirkabir University of Technology, Computer Engineering Department, Tehran, Iran
Email: {msaeedi, szamani, msedighi}@ aut.ac.ir

Keywords

Emerging Technologies, Reversible logic synthesis, reversible computing and quantum computing

Abstract

Reversible computing plays an important role in various research areas including quantum computing. Among open research problems, reversible circuit synthesis has recently received significant attention. In this paper, we propose a new non-search based forward-looking synthesis (FLS) algorithm for reversible circuits. Compared with the widely used search-based methods, FLS is guaranteed to produce a result and can lead to a solution with much fewer steps. Next, a forward-looking order independent synthesis (FLOPS) algorithm is introduced which uses several novel criteria to improve the FLS resulted cost. To evaluate the proposed algorithms, different circuits taken from the literature are used. The experimental results corroborate the expected findings.

1. INTRODUCTION

A fully-specified Boolean specification with the same number of inputs and outputs is called reversible if it maps each input assignment to a unique output assignment. As a result, the truth table of a reversible specification of size *n* can be defined as a set of integers *{0, 1, . . . , 2^n - 1}* probably with different order. Figure 1 shows a possible reversible specification of size 3 with its integer representation.

a_1	a_2	a_3	f_1	f_2	f_3	*Representation*
0	0	0	0	1	0	2
0	0	1	1	1	1	7
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	6
1	0	1	0	1	1	3
1	1	0	1	0	0	4
1	1	1	1	0	1	5

Figure 1- A possible reversible specification of size 3, shaded box shows the integer representation of 2nd minterm.

Landauer [1] proved that using conventional irreversible logic gates leads to energy dissipation, regardless of the underlying circuit. Bennett [2] stated that to avoid power dissipation in a circuit, the circuit must be built with reversible gates. Today, reversible logic has received considerable attention in various research areas such as low power CMOS design [3], optical computing [4], quantum computing [5].

Reversible logic synthesis is defined as the ability to automatically generate a reversible circuit from a given reversible specification. The synthesis of reversible circuits is significantly more complex than the synthesis of traditional irreversible gates [6] and it is one of the most recent research problems.

In order to synthesize reversible circuits, various synthesis algorithms have been proposed yet. Among different algorithms, search-based methods [7], [8] have attracted more attentions where an extensive exploration is required. In other words, the time complexity of these algorithms limits their applications to small and medium size circuits.

In this paper, a non-search based synthesis method for reversible circuits is proposed which uses a forward-looking methodology to reach a result. The rest of this paper is organized as follows: in Section 2, basic concepts are presented. Previous work on reversible logic synthesis is reviewed in Section 3. Our forward-looking synthesis algorithm is presented in Section 4 and improved in Section 5. Experimental results are reported in Section 6 and finally, Section 7 concludes the paper.

2. Basic Concept

An *n*-input, *n*-output gate is called reversible if it realizes a reversible function. Previously, various reversible gates with different functionalities have been proposed [9], [10] and [11]. Among them, CNOT-based gates comprise an important class of reversible gates [12]-[20] which are also considered in this paper and denoted as follows:

Definition 1: An *n*-input, *n*-output CNOT gate $CNOT_n(x_1,x_2,\ldots,x_n)$ passes the first *n-1* lines unchanged. These lines are referred to control lines. This gate flips the n^{th} line if the control lines are all one. In other words, we have: $x_{i(out)}=x_i$ $(i<n)$, $x_{n(out)}=x_1x_2\ldots x_{n-1}\oplus x_n$. Some authors (see [14]) assume that complementation can also be internal to a CNOT-based gate. Therefore, it is possible to have a $CNOT_3(a',b',c)$ gate to refer to $c(new)=c\oplus a'b'$,

a(new)=a and *b(new)=b*. This assumption is also considered in this paper.

In the following section, previous algorithms for reversible circuit synthesis are reviewed.

3. Previous Work

Several algorithms have recently been proposed to synthesize a reversible circuit. Toffoli in [11] presented an algorithm to implement a function using CNOT-based gates. In [12], a new incremental approach was presented which uses shared binary decision diagrams for representing a reversible specification and measuring circuit complexity. The proposed algorithm selects reversible gates based on the complexity of the rest of logic. Some authors used transformation-based algorithms for reversible circuit synthesis [13]-[15]. However, these algorithms usually use local transformations to optimize the results of other algorithms.

Shende et al. [16] investigated a number of techniques to synthesize optimal and near-optimal reversible circuits that require little or no temporary storage. They also provided some properties about even and odd permutation functions. The authors of this paper [17] investigate the algebraic characterizations of CNOT-based reversible circuits to propose a new methodology to synthesize CNOT-based quantum circuits [18].

As the size of a reversible circuit can be large, a practical algorithm for reversible circuit synthesis may become extremely difficult. Due to the lack of a systematic method, search-based algorithms are widely used for reversible circuit synthesis where an extensive exploration is required to find a possible implementation of the circuit. This method is used in [7] to propose a synthesis algorithm for reversible circuits. This algorithm is further improved by the authors of this paper to reach a result within fewer node searches [8]. However, as search-based algorithms evaluate all possible gates to find an implementation of the circuit, they cannot be used to synthesize large functions.

Miller et al. in [20] considered the use of Rademacher-Walsh spectral techniques in order to guide the search process. In other words, a defined cost function is used to select the best possible candidate. However, because of the evaluation of all possible gates, their work is also limited to small circuits.

In the following section, we propose a non-search based synthesis algorithm for reversible circuits which uses a Forward-looking strategy to produce a solution for a given specification after several steps.

4. Forward-Looking Synthesis Algorithm

In order to propose our forward-looking synthesis (FLS) algorithm, several definitions and theorems are required. In this paper, the i^{th} input (output) variable is denoted as a_i (f_i). In addition, a general reversible specification of size n is shown as $F(a_1,a_2,\ldots,a_n)=(f_1,f_2,\ldots,f_n)$.

Assume that a set of CNOT-based gates $(g_1,g_2,\ldots g_k)$ is used to produce a reversible function f_i $(i=1,..,n)$ from its corresponding a_i as shown in Figure 2. Since the circuit is reversible, one can use the same set of gates in reverse order, i.e. $(g_k,g_{k-1},\ldots g_1)$, to produce a_i from f_i. This fact is used in our synthesis algorithm as stated later.

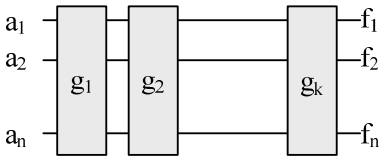

Figure 2- Producing n reversible functions from k reversible gates

Definition 2: The application of a reversible $CNOT_n(x_1,x_2,\ldots,x_n)$ gate at the output side of a reversible specification F is called *output translation*. Therefore, after using several output translations each reversible function f_i will be transformed to its corresponding a_i.

As each output translation is a reversible $CNOT_k(x_1,x_2,\ldots,x_k)$ $k\leq n$ gate, the result of using an output translation will also be reversible. Furthermore, it can be easily verified that only one function (i.e. f_k) is changed at a time after using an output translation. Lemma 1 explains the results of using an output translation on a given reversible specification F.

Lemma 1: (a) An output translation for a given specification F, exchanges the location of 2^k $(k\leq n-1)$ minterm pairs. (b) Conversely, exchanging the location of 2^{k-1} $(k=n-m+1)$ minterm pairs with the following properties:

- all of the 2^k $(k=n-m+1)$ minterms have the same value on their $m-1$ specific bit locations.
- the two minterms of each pair differ only in one bit position.

has the same result as applying an output translation $CNOT_m\left(f_{idx_1}, f_{idx_2},\ldots, f_{idx_{m-1}}, f_{idx_m}\right), idx_k \in (1..n), m\leq n$ to F.

Proof: (Case a): Assume that an output translation $CNOT_m\left(f_{idx_1}, f_{idx_2},\ldots, f_{idx_{m-1}}, f_{idx_m}\right)$ *where* $idx_k \in (1..n), m\leq n$ is applied to F. It can be easily checked that using this translation changes f_{idx_m} to $f_{idx_1} f_{idx_2}\ldots f_{idx_{m-1}} \oplus f_{idx_m}$. It is important to note that f_{idx_k} for $k\in(1..m-1)$ can also be a complemented function. As for 2^k $(k=n-m+1)$ minterms, we have $f_{idx_1} f_{idx_2} f_{idx_{m-1}} = 1$, by using this translation the location of 2^k minterms are changed. Furthermore, as an XOR gate is applied to f_{idx_m}, the

locations of all 2^{k-1} minterm pairs $m_i : f_{idx_m} = 1$ and $m_j : f_{idx_m} = 0$ will be exchanged.

(Case b): Since there are 2^k $(k=n-m+1)$ minterms which have the same value on their $m-1$ bits, there are 2^{k-1} minterm pairs each of which differs only in one bit position. Therefore, exchanging their locations has the same effect as applying an output translation $CNOT_m\left(f_{idx_1}, f_{idx_2}, \ldots, f_{idx_{m-1}}, f_{idx_m}\right), idx_k \in (1..n), m \le n$ □

It can be said that the goal of our FLS algorithm is to generate a set of output translations with a specific order which when applied to the reversible specification F, generates a_i from f_i. Figure 3 shows our synthesis algorithm.

Algorithm FLS

Input: A reversible specification $F(a1, a2, \ldots, a_n) = (f_1, f_2, \ldots, f_n)$
Output: A set of reversible CNOT-based gates which when applied to F produces an identity function.
Notation: The i^{th} function (variable) of j^{th} minterm is denoted as $f_{i,mj}$ $(a_{i,mj})$. Consequently, The i^{th} minterm of j^{th} function (variable) is denoted as $m_{i,fj}$ $(m_{i,aj})$

```
i = 1;
repeat
  reset all of the 2^n minterms to be unvisited.
  for each minterm m_j (j = 1 ... 2^n) do                    FLS core
    if m_j is not visited then
      if f_{i,mj} ≠ a_{i,mj} then
      begin
        mark the minterm m_{j,fi} as a visited minterm
        select the minterm m_{k,fi} which differs from m_{j,fi} in its i^th variable
        if m_{k,fi} is below m_{j,fi} then
          exchange the locations of m_{j,fi} and m_{k,fi}. (Therefore f_{i,mj} = a_{i,mj})
          mark the minterm m_{k,fi} as a visited minterm
        else if m_{k,fi} is above than m_{j,fi} then
          if f_{p,mk} ≠ a_{p,mk} (p=1...n) for at least one p then
            exchange the locations of m_{j,fi} and m_{k,fi}. (Therefore f_{i,mj} = a_{i,mj})
            mark the minterm m_{k,fi} as a visited minterm
      end
  i = (i+1) mod n;
until f_i = a_i for each i ∈ (1,..,n)
```

Figure 3- Our FLS synthesis algorithm

It is worthwhile to note that at each step, the proposed FLS algorithm converts the current minterm of a reversible function to its corresponding input variable regardless of the previous movements. In other words, the algorithm uses a forward-looking strategy to reach a result. Consider the following example for more details:

Example 1: (Example 2 from [7]) Consider a reversible specification $F(a_1,a_2,a_3)=(7,0,1,2,3,4,5,6)$ defined as the first and the second columns of Figure 4. In this figure, similar shaded boxes are used to represent the corresponding minterms before and after each output translation.

Step 1: Select the first variable (i.e. a_1). Start with $j=1$ and check the first minterm of f_1 (i.e. 1) and a_1 (i.e. 0). As these minterms are not equal, the 5th minterm of F (i.e. *011*) should be selected. Note that this minterm and the first one differ only in their first variable. Furthermore, the minterm *011* (the 5th minterm) is below *111* (1st) in F. Therefore, these two minterms are exchanged. Then, other minterms of f_1 (i.e. $j=2,3,4,6$ and *7*) are left unchanged. The third column in Figure 4 shows the specification of F after this translation.

Step 2: Select the second variable (i.e. a_2). Start with $j=1$ and check the first minterm of f_2 (i.e. 1) and a_2 (i.e. 0). As these minterms are not equal, the 3rd minterm of F (i.e. *001*) should be selected. Note that these two minterms differ only in their second variable. Furthermore, the minterm *001* (3rd minterm) is below *011* (1st) in F. Therefore, these two minterms are exchanged. Then, the second and the forth minterms of f_2 are equal to their corresponding a_2 minterms. However, for $j=5$, the second minterm of f_2 (i.e. 1) and a_2 (i.e. 0) are not equal. Therefore, the minterms *101* (the 5th minterm) and *111* (the 7th minterm) are exchanged. Finally, the 6th and the 8th minterms of f_2 are equal to their corresponding a_2 minterms. The forth column in Figure 4 shows the specification of F after this translation.

			F			Step1			Step2			Step3		
a_1	a_2	a_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
0	0	0	1	1	1	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**
0	0	1	0	0	0	0	0	0	0	0	0	**0**	**0**	**1**
0	1	0	0	0	1	0	0	1	**0**	**1**	**1**	**0**	**1**	**0**
0	1	1	0	1	0	0	1	0	0	1	0	**0**	**1**	**1**
1	0	0	0	1	1	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
1	0	1	1	0	0	1	0	0	1	0	0	**1**	**0**	**1**
1	1	0	1	0	1	1	0	1	**1**	**1**	**1**	**1**	**1**	**0**
1	1	1	1	1	0	1	1	0	1	1	0	**1**	**1**	**1**

Figure 4- The specification of Example 1 before and after three translations.

Step 3: Select the third variable (i.e. a_3). Based on the previous two steps and the proposed algorithm, it can be easily checked that the locations of all four minterm pairs (i.e. 1 & 2, 3 & 4, 5 & 6 and 7 & 8) should be exchanged. Since after the third translation, we have $f_i=a_i$ for each $i\in(1\ldots n)$, the algorithm is finished. The fifth column in Figure 4 shows the specification of F after this translation.

In order to find the CNOT-based implementation of each output translation, one can use the results of its previous translation to find each gate. Figure 5 shows the implementation of each output translation and the final circuit. □

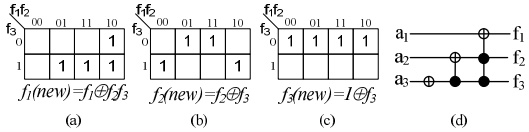

Figure 5- The CNOT-based implementation of each output translation for (a) Step 1, (b) Step 2, (c) Step 3 and (d) the final circuit

As each output translation changes only one f_i $(i=1\ldots n)$, if the previous output translation placed the minterms of the i^{th} variable at their right locations, the current output translation applied to another variable would not change their locations. In other words, the

successor output translations do not destroy the predecessor results. Furthermore, it is important to note that based on Lemma 1, each output translation has an equivalent CNOT-based implementation. Therefore, the result of applying the proposed FLS algorithm is a set of CNOT-based gates which should be applied in reverse order to the variables $(a_1,a_2,\ldots,a_n)$ to produce the reversible functions $(f_1,f_2,\ldots,f_n)$.

Theorem 1: The FLS algorithm will converge to a possible implementation after several steps.

Proof: Consider a reversible specification F of size n. Assume that after the i^{th} step, several minterms which are represented as a set Σ, are placed at their right positions and in the $(i+1)^{th}$ step, the algorithm works on the k^{th} variable $(k\leq n)$. Suppose that the k^{th} variable of a minterm, i.e. the m^{th} ($m\notin\Sigma$) minterm, is not correct. Accordingly, the algorithm finds a minterm placed at location p which differs from the m^{th} minterm only in its k^{th} variable. If $p\in\Sigma$ and $p<m$, the algorithm does nothing to avoid instability in minterm locations. However, as the m^{th} minterm is placed at a wrong position (for example, the position of the q^{th} minterm, $q\notin\Sigma$), there must be another minterm, i.e. the q^{th} minterm, which should be exchanged with the m^{th} minterm during the next steps. Therefore, the algorithm does not finish at the current step and the algorithm will reach the other cases, i.e. $p\notin\Sigma$ or $p\in\Sigma$ *and* $p>m$). For these cases, the algorithm exchanges the location of the p^{th} minterm with that of the m^{th} minterm. Then, the k^{th} variable of the m^{th} minterm will be correct and the algorithm moves forward to check other minterms. As each output translation does not change the results of the previous ones, the algorithm will gradually place all minterms at their right positions. Therefore, the algorithm will lead to a valid result. □

By the previous theorem, we show that the proposed FLS algorithm converges after several steps. However, it may be possible to reduce the number of steps (i.e. cost) as shown in the following section.

5. Forward-Looking Order indePendent Synthesis (FLOPS) Algorithm

As shown in Figure 3, our FLS algorithm always begins with the first minterm of the first variable. Then, the second minterm is selected and this process is continued to reach the 2^nth minterm. Again, the second variable is chosen and this process is repeated several times until all 2^n minterms of the n output variables become the same as their equivalent input variables. Theorem 1 shows that the proposed FLS algorithm always converges to a result. However, the resulted cost may be very high.

In this section, we propose another forward-looking order indePendent synthesis (FLOPS) algorithm which considers at most n output translations to select the possible gate at each step of the algorithm. For the following paragraphs, the minterms of input variables are referred as input minterms (IM). Similar notation is used for output minterms (OM). In order to propose the FLOPS algorithm, the following definitions are required:

Definition 3: The *distance* of the k^{th} output minterm D_k^{OM} is defined as $\sum_{i=1}^{n}(b_i^{IM}\oplus b_i^{OM})$ where b_i^{OM} and b_i^{IM} represent the i^{th} bit of the k^{th} output and input minterms, respectively and n is the size of the circuit.

Definition 4: The maximum, minimum and total distance of a reversible specification are defined as $D_{\max}=\max_{k=1}^{2^n}(D_k^{OM})$, $D_{\min}=\min_{k=1}^{2^n}(D_k^{OM})$ and $D_{Total}=\sum_{k=1}^{2^n}D_k^{OM}$, respectively.

Based on Definition 3, Definition 4 and Figure 1, it can be verified that $D_1^{OM}=1$, $D_2^{OM}=2$, $D_3^{OM}=1$ $D_4^{OM}=1$, $D_5^{OM}=1$, $D_6^{OM}=2$, $D_7^{OM}=1$and $D_8^{OM}=1$. We also have $D_{min}=1$, $D_{max}=2$ and $D_{Total}=10$. Figure 6 shows the FLOPS algorithm.

Algorithm FLOPS

Input: A reversible specification $F(a_1, a_2, \ldots, a_n) = (f_1, f_2, \ldots, f_n)$
Output: A set of reversible CNOT-based gates producing an identity function when applied to F.

```
Repeat
  for each output function f_i (i=1...n) do
    if f_i has not been considered previously then
    begin
      run the FLS core temporarily (Figure 3)
      calculate D_max(fi), D_min(fi) and D_Total(fi).
    end
  select the function f_idx with minimum D_Total.
  if more than one exist then select the minimum D_max function
  if more than one exist then select the minimum D_min function
  if more than one exist then select the first one.
  assign i=idx
  run the FLS core permanently (Figure 3)
until D_p=0 for each p∈(1...n)
```

Figure 6- The FLOPS synthesis algorithm

As shown in Figure 6, the FLOPS algorithm does not work on the output functions based on a predefined order as the FLS method. In other words, FLOPS tries to select the best possible function which results in further total distance reduction. If more than one function exists, the algorithm selects the one with the maximum D_{max} reduction. Similar selection criteria are also considered for the other cases as shown in Figure 6. It is important to note that the proposed selection metrics of the FLOPS algorithm have no effects on the result of Theorem 1. Therefore, it can be verified that the FLOPS algorithm also converges to a CNOT-based implementation.

As the synthesized result of the FLS algorithm for the specification of Example 1 is optimized, the FLOPS algorithm cannot produce a lower cost circuit for this case. Consider the following example for more details:

Example 2 (Example 1 from [7]): A reversible specification $F(a_1,a_2,a_3)=(1,0,3,2,5,7,4,6)$ is given. Figure 7 and Figure 8 show the results of the FLOPS

and FLS algorithms, respectively. As shown in these figures, only exchanged minterms are reported to save space. We also demonstrated the resulted CNOT-based gates in these figures. It is clear that the FLOPS algorithm reach a better circuit in fewer steps.

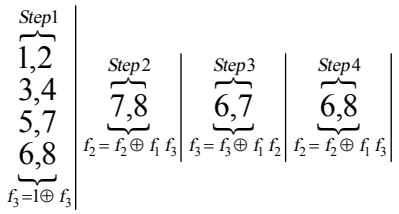

Figure 7- The results of running the FLOPS algorithm on the specification of Example 2.

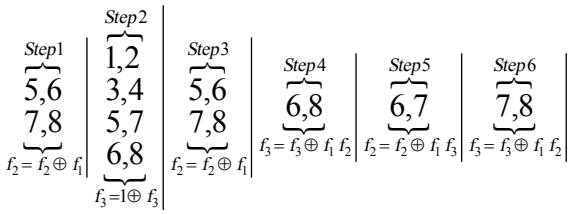

Figure 8- The results of running the FLS algorithm on the specification of Example 2.

In order to compare the time complexity of our proposed FLS approach with the search based methods, assume that a possible CNOT-based implementation of a reversible specification *F* of size *n* needs at most *h* gates. It can be verified that $n\times2^{n-1}$ possible gates must be evaluated to simplify *F* at each step of a search-based method [7], [8]. As a result, a search-based algorithm evaluates $h\times n\times 2^{n-1}$ gates in the worst case. On the other hand, the proposed FLS methods consider *h* output translations, i.e. gates, in the worst case. Therefore, it can be said that the time complexity of the search-based methods are much higher than that of the proposed non-search based FLS method by $n\times 2^n$.

On the other hand, as the proposed FLOPS algorithm considers at most *n* possible output translations at each step, its time complexity will be O(*h*×*n*). Therefore, FLOPS algorithm is more efficient than the search-based methods by 2^n. In the following section, the experimental results are shown.

6. Experimental Results

Our proposed algorithms were implemented in C++. To save space, only exchanged minterm numbers are stated for each step separated by a '{}' symbol. Therefore, Example 1 (Figure 4) is shown as {1&5}{1&3,5&7}{1&2,3&4,5&6,7&8}. In addition, only CNOT gate control and target lines are maintained. For example, we use *(a,b,c)* to refer to $CNOT_3(a,b,c)$. To evaluate the proposed method, we use the first eight examples of [7] and five new ones which cannot be synthesized by the method of [7]. The results of using our methods are shown in Table 1. As shown in this table, the proposed FLOPS algorithm not only has the ability to produce a result for all of the attempted specifications but also can reach a cost-effective result with much fewer steps. The proposed output translations and the final circuit of each specification are also shown in Table 2.

Table 1- The results of using the proposed synthesis methods compared with two recent methods [7][8]

Circuit #	Reversible Specification	FLS	FLOPS	[7]	[8]
		Number of Steps		Number of Searched Nodes	
1	(1,0,3,2,5,7,4,6)	6	**4**	11	15
2	(7,0,1,2,3,4,5,6)	3	**3**	761	300
3	(0,1,2,3,4,6,5,7)	3	**3**	7	10
4	(0,1,2,4,3,5,6,7)	7	**5**	156	786
5	(0,1,2,3,4,5,6,8,7,9, 10,11,12,13,14,15)	15	**7**	1001	696
6	(1,2,3,4,5,6,7,0)	3	**3**	4	4
7	(1,2,3,4,5,6,7,8,9,10, 11,12,13,14,15,0)	4	**4**	5	5
8	(0,7,6,9,4,11,10,13, 8,15,14,1,12,3,2,5)	5	**5**	230	139
9	(3,6,2,5,7,1,0,4)	**8**	9	-	368
10	(1,2,7,5,6,3,0,4)	8	**8**	-	141
11	(4,3,0,2,7,5,6,1)	8	**7**	-	6291
12	(7,5,2,4,6,1,0,3)	6	**6**	-	2092
13	(6,2,14,13,3,11,10, 7,0,5,8,1,15,12,4,9)	23	**14**	-	8570

7. Conclusions

In this paper, a new forward-looking order independent non-search based synthesis algorithm was proposed which requires a few steps to synthesize a given reversible specification. In order to evaluate the algorithm, we used eight examples taken from the literature and five new ones that cannot be synthesized by one of the most recent search-based methods [7] or needs many steps to find a result [8]. It was shown that the proposed algorithm could lead to a result for all of the circuits very fast.

8. References

[1] R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM Journal*, vol. 5, pp. 183-191, July 1961.

[2] C. Bennett, "Logical Reversibility of Computation," *IBM Journal*, vol. 17(6), pp. 525-532, November 1973.

[3] G. Schrom, "Ultra-Low-Power CMOS Technology," PhD Thesis, Technischen Universitat Wien, June 1998.

[4] E. Knill, R. Laamme, and G. J. Milburn, "A Scheme for Efficient Quantum Computation with Linear Optics," *Nature*, pp. 46-52, January 2001.

[5] M. Nielsen and I. Chuang, "Quantum Computation and Quantum Information" Cambridge University Press, 2000.

[6] A. Mishchenko and M. Perkowski, "Logic synthesis of Reversible Wave Cascades," *IWLS*, 2002, pp. 197-202.

Table 2- The synthesized results of our FLOPS algorithm for the attempted examples

Circuit #		Output Translations	The synthesized circuits
[7]	1	See Figure 7	See Figure 7
	2	{1&8, 2&3,4&5,6&7}{1&7,3&5}{1&5}	$(f_3),(f_1',f_2),(f_2',f_3',f_1)$
	3	{6&8}{6&7}{7&8}	$(f_1,f_2,f_3),(f_1,f_3,f_2),(f_1,f_2,f_3)$
	4	{4&7}{5&8}{4&5}{4&8}{5&7}	$(f_1,f_3',f_2),(f_2,f_3,f_1),(f_1,f_2,f_3),(f_2,f_3,f_1),(f_1,f_3',f_2)$
	5	{8&9}{8&15}{8&16}{8&9}{9&16}{9&15}{9&13}	$(f_1,f_3',f_4',f_2),(f_1,f_2,f_4',f_3),(f_1,f_2,f_3,f_4),$ $(f_2,f_3,f_4,f_1),(f_1,f_2,f_3,f_4),(f_1,f_2,f_4',f_3),(f_1,f_3',f_4',f_2)$
	6	{1&8,2&3,4&5,6&7}{2&8,4&6}{4&8}	$(f_3),(f_3,f_2),(f_2,f_3,f_1)$
	7	{1&16,2&3,4&5,6&7,8&9,10&11,12&13,14&15}{2&16,4&6,8&10,12&14} {4&16,8&12}{8&16}	$(f_4),(f_4,f_3),(f_3,f_4,f_2),(f_2,f_3,f_4,f_1)$
	8	{2&14,3&15,6&10,7&11}{2&12,4&10,6&8,14&16}{4&12,8&16}{7&15}{6&14}	$(f_3,f_2),(f_4,f_3),(f_3,f_4,f_1),(f_2,f_3,f_4',f_1),(f_2,f_3',f_4,f_1)$
9		{1&6,4&5}{2&8}{2&7}{4&6}{5&7,6&8}{1&2}{6&8}{6&7}{7&8}	$(f_3,f_2)(f_1,f_3',f_2),(f_2',f_3',f_1),(f_2,f_3,f_1),$ $(f_1,f_3),(f_1',f_2',f_3),(f_1,f_2,f_3),(f_1,f_3,f_2),(f_1,f_2,f_3)$
10		{1&6,2&7,3&4,5&8}{1&4,3&6}{7&8}{1&7}{2&3}{1&8}{1&3}{7&8}	$(f_2),(f_3,f_1),(f_2,f_3',f_1),(f_1,f_2,f_3),(f_1',f_2',f_3),$ $(f_2,f_3',f_1),(f_1',f_3',f_2),(f_1,f_2,f_3)$
11		{2&8,3&4}{5&6}{2&5}{1&2,6&8}{1&4}{4&6}{2&6}	$(f_1',f_2),(f_1,f_3,f_2),(f_1,f_2',f_3),(f_3,f_1),$ $(f_1',f_2',f_3),(f_1',f_3,f_2),(f_3,f_2',f_1)$
12		{1&8,2&6}{4&7}{1&2}{5&7}{1&4}{2&4}	$(f_3,f_1),(f_2',f_3',f_1),(f_1',f_3,f_2),(f_1,f_3',f_2),$ $(f_1',f_2',f_3),(f_1',f_3,f_2)$
13		{1&15}{2&9,5&12}{4&13,6&16}{1&2}{3&7,4&16}{5&10,6&13}{3&9,4&12} {5&6,10&13}{7&15}{2&6,5&14}{2&13}{6&13}{5&13}{9&11}	$(f_1',f_2,f_4',f_3),(f_1',f_2',f_3),(f_1,f_4,f_3),$ $(f_1',f_3',f_4',f_2),(f_1,f_3,f_2),(f_3',f_4,f_2),$ $(f_2',f_3,f_1),(f_3',f_4,f_1),(f_2,f_3,f_4',f_1),$ $(f_2,f_3',f_4),(f_1',f_3',f_4),(f_1',f_2,f_3',f_4),$ $(f_2,f_3',f_4',f_1),(f_1,f_2',f_4',f_3)$

[7] P. Gupta, A. Agrawal, and N. K Jha, "An Algorithm for Synthesis of Reversible Logic Circuits," *TCAD*, November 2006.

[8] M. Saeedi, M. Saheb Zamani, M. Sedighi, "On the Behavior of Substitution-Based Reversible Circuit Synthesis Algorithms: Investigation and Improvement", *ISVLSI*, 2007.

[9] R. Feynman. "Quantum Mechanical Computers," *Optic News*, 11:11-20, 1985.

[10] E. Fredkin, and T. Toffoli, "Conservative Logic," *International Journal of Theoretical Physics*, 21:219-253, 1982.

[11] T. Toffoli. "Reversible computing," Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci, 1980.

[12] P. Kerntopf, "A New Heuristic Algorithm for Reversible Logic Synthesis," *DAC*, pp. 834-837, 2004.

[13] K. Iwama, and Y. Kambayashi, and S. Yamashita, "Transformation Rules for Designing CNOT-Based Quantum Circuits," *DAC*, pp.419-424, 2002.

[14] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck, "Quantum Circuit Simplification Using Templates," *DATE*, pp. 1208-1213, 2005.

[15] V. V. Shende, A. K. Prasad, K. N. Patel, I. L. Markov and J. P. Hayes, "Scalable Simplification of Reversible Circuits," *IWLS*, 2003.

[16] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of Reversible Logic Circuits," *TCAD*, vol. 22(6), pp. 710-722, 2003.

[17] M. Saeedi, M. Saheb Zamani, M. Sedighi, "Algebraic Characterization of CNOT-Based Quantum Circuits with its Applications on Logic Synthesis", *DSD*, 2007.

[18] M. Saeedi, M. Sedighi, M. Saheb Zamani, "A New Methodology for Quantum Circuit Synthesis: CNOT-Based Circuits as an Example", *IWLS*, 2007.

[19] D. M. Miller, D. Maslov, and G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis," *DAC*, pp. 318-323, 2003.

[20] D. M. Miller, G. W. Dueck, "Spectral Techniques for Reversible Logic Synthesis," *RM*, 2003.