

Logic Synthesis with Cascades of New Reversible Gate Families

Mozammel H A Khan and Marek A Perkowski*

Computer Science and Engineering Department, East West University, 45 Mohakhali C/A, Dhaka 1212, BANGLADESH. mhakistan@ewubd.edu, mhakistan@accessstel.net

*Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Gusong-dong, Yusong-gu, Daejeon 305-701, KOREA. mperkows@ee.kaist.ac.kr

Abstract: Reversible circuits are currently one of top approaches to power minimization and they are the approach of a growing importance. In this paper, the well known Feynman gate is generalized to a $k*k$ gate and a new generalized $k*k$ reversible gate family is proposed. A synthesis method for multi-output (factorized) SOP using cascades of the new gate family is presented for the first time. For utilizing the benefit of sub-expression sharing among the SOPs, two graph-based data structures – connectivity tree and implementation graph are used. Experimental results with some MCNC benchmark functions show that the number of gates in the multi-output SOP cascades is almost equal to the number of products in the multi-output SOP. However, this cascaded realization of multi-output SOP generates a large number of garbage outputs and requires large number of input constants, which need to be reduced in the future research. Another synthesis method for EXOR of (factorized) SOPs using cascades of the new gate family and generalized Feynman gate is also introduced for the first time. The usefulness of the new gate family to synthesize single-output (factorized) ESOP cascades is also briefly outlined. Design of full-adder circuit with two new gates and one Feynman gate is presented, which is more efficient than the designs of [18], [21], and [25] in terms of gate count. All these synthesis methods are technology-independent and can be used in association with any known or future reversible technology.

1. Introduction

Motivation to Study Reversible Logic

Landauer [1] showed that a computational system built using traditional irreversible logic gates such as AND or multiplexer leads inevitably to energy dissipation, regardless of the technology used to realize the gates. The energy loss due to irreversible gates is negligible for current silicon technologies using adiabatic design. However, it is well known that Moore's Law will stop to function around years 2010 – 2020 and some dramatic changes will therefore have to happen in microelectronics not later than the middle of this century [2]. In this time reversible design will become of primary importance.

Bennett [3] showed that for power not to be dissipated in an arbitrary circuit, it is necessary that the circuit be built from *reversible gates*. In principle, reversible logic gates dissipate arbitrary little heat and the use of reversible operations is likely to become more attractive. It should be noted that Bennett's theorem is only a necessary but not sufficient condition. Its extreme importance lies in the technological necessity that *every future technology will have to use reversible gates in order to reduce power*. Quantum logic is reversible and the problem of efficient designs of quantum circuits [4] includes as its sub-problem the problem of synthesis using classical reversible gates. Therefore, many of the methods for reversible logic synthesis can be adapted to quantum logic as well.

Reversible Logic

Reversible logic are circuits (gates) that have the same number of inputs and outputs and have one-to-one mappings between input vectors and output vectors; thus the input vector states can be always uniquely reconstructed from the output vector states. A reversible gate with k inputs and k outputs is called a $k*k$ gate. All gates in a reversible circuit must be reversible. A *balanced function* has half of minterms with value 1 and half with value 0. A reversible circuit without constants on inputs realizes a balanced function on each of its outputs. Therefore, it can realize non-balanced functions only with *garbage (unused)* outputs and constant values on its primary inputs.

As the truly low-power circuits (with power arbitrarily small) cannot be built without the concept of reversible logic, various technologies for reversible logic are recently intensively studied. These technologies include: (i) standard CMOS [5, 6], (ii) optical technologies [7, 8], (iii) quantum logic technologies [9, 10], (iv) DNA technology [8], and (v) mechanical technology (nanotechnology) [11].

Many binary reversible logic gates have been proposed [5, 12-16]. There exists only one 1*1 gate, which is an *inverter* (a wire is not considered as a gate). This gate is shown in Figure 1(a) and is very important since it does not introduce garbage outputs. There are several 2*2 gates in reversible logic and they are all linear. A gate is *linear* when all its outputs are linear functions of input variables. The 2*2 *Feynman gate* (also called *controlled-not* or *quantum EXOR* or *reversible EXOR*) is shown in Figure 1(b). This gate is a *one-through gate*, which means that one of its input variables is also an output. When $A=0$ then $Q=B$; when $A=1$ then $Q=B'$. With $B=0$ the 2*2 Feynman gate is used as a *fan-out* (or *copying*) *gate*. Every linear reversible function can be built by using only 2*2 Feynman gates and inverters. There exist $8! = 40,320$ 3*3 reversible logic gates, some of them with interesting properties. Two of the universal 3*3 reversible gates have been much studied: *Fredkin gate* [12] and *Toffoli gate* (also called *3*3 Feynman gate* or *Controlled-controlled-not*). The 3*3 *Fredkin gate* is shown in Figure 1(c). In terms of classical logic, this gate is just two multiplexers controlled in a flipped (permuted) way from the same control input A . Fredkin gate is also a one-through gate. The 3*3 *Toffoli gate* is shown in Figure 1(d). Toffoli gate is a *two-through gate*, because two of its inputs are returned unmodified as its outputs. When $C=0$ then $R=AB$, so AND gate is realized on R output. When $A=1$ then $R=B \oplus C$, so EXOR gate is realized on R output. When $B=1$ then $R=A \oplus C$, so EXOR gate is realized on R output. The 3*3 *Kerntopf gate* [13, 16, 17] is shown in Figure 1(e). When $C=1$ then $P=A+B$, $Q=AB$, $R=B'$, so OR and AND gates are realized on outputs P and Q , respectively, with C as the controlling input value. When $C=0$ then $P=A'B'$, $Q=A+B'$, $R=A \oplus B$. Therefore, for control input value 0, the gate realizes NOR, IMPLICATION, and EXOR on its outputs P , Q , and R , respectively. The 3*3 Kerntopf gate is not a one-through nor a two-through gate. Generalized $n*n$ Feynman, Fredkin, Toffoli, and Kerntopf gates are introduced in [18].

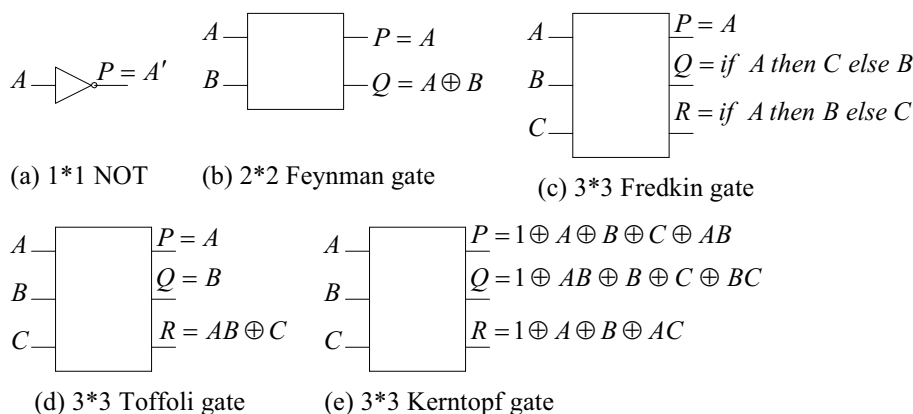


Figure 1: Some useful binary reversible gates.

Reversible Logic Synthesis

Reversible logic synthesis is more difficult than classical logic synthesis. The major constraints of reversible logic synthesis are: (i) the fan-out of every signal, including primary inputs, is one, (ii) the graph of the reversible circuit must be a DAG (directed acyclic graph), which means that there must be no loops of gates or internal loops in a gate, and (iii) many of the practical functions are not themselves reversible and need to be made reversible before implementing them with reversible gates. Systematic logic synthesis algorithms for reversible logic are still very immature, but some methods have been proposed [15, 17-22]. Most of the papers discuss design using Feynman, Toffoli, and Fredkin gates. In [21] compositional synthesis methods for reversible logic have been presented. The simplest structure for composition is cascades. Reversible logic synthesis using cascades of gates is presented in [15, 18, 22]. The cascades have the same number of

intermediate signals at every level. When a gate is applied for composition it subtracts as many input variables as it has inputs and it adds the same number of new variables to the next level. The reversible cascades are recently extensively studied, as many well-known standard logic methods can be adapted to reversible cascades.

The objectives of a good synthesis algorithm for reversible logic are: (i) not to create excessive garbage outputs, (ii) not to create unnecessary input constants, (iii) to avoid leading output signals of gates to more than one input, as such fan-out requires additional copying gate, (iv) to minimize the length of the circuit to reduce the delay of the circuit, and (v) to minimize the total number of gates.

Contribution of This Paper

The goal of this paper is to develop new logic design methods based on cascades of new reversible gate families and respective synthesis algorithms for them. For this purpose, a new generalization of $k*k$ Feynman gate and a new generalized $k*k$ gate family are proposed. For the first time, to our knowledge, synthesis methods for multi-output (factorized) SOP and EXOR of (factorized) SOPs using cascades of new reversible gate families are presented in this paper. Moreover, synthesis method for single-output (factorized) ESOP is also presented for this kind of new cascades. Design of full-adder circuit using two new gates and one Feynman gate is also presented, which is more efficient than the designs of [18], [21], and [25] in terms of gate count. For all these synthesis methods two graph-based data structures – connectivity tree and implementation graph are used. Experimental results with some MCNC benchmark functions show that the number of gates in most of the multi-output SOP cascades is almost equal to the number of products in the multi-output SOP. However, this cascaded realization of multi-output SOP generates a large number of garbage outputs. All these synthesis methods are technology-independent and can be used in association with any known or future reversible technology.

2. New Families of $k*k$ Reversible Gates

2.1 New generalization of Feynman gate

One of our goals in this paper is to formulate a logic synthesis method for EXOR of any number of (factorized) SOPs, where the (factorized) SOPs will be EXOR-summed by a reversible gate. For this purpose, we proposed a new $k*k$ generalization of Feynman gate in Figure 2(a). The gate is a $(k - 1)$ - through gate and the output P_k produces the EXOR-sum of the input variables. By Theorem 1, it can be shown that the gate is a reversible gate.

Theorem 1: $k*k$ Feynman gate described by the equations $\{P_1 = A_1, P_2 = A_2, \dots, P_{k-1} = A_{k-1}, P_k = A_1 \oplus A_2 \oplus \dots \oplus A_{k-1} \oplus A_k\}$ is a reversible gate.

Proof. As the gate has k inputs, the truth table of the gate will have 2^k rows. The most significant $k - 1$ outputs of the gate are pass-through outputs. So, each of the 2^{k-1} possible combinations of these $k - 1$ most significant output bits will appear in pairs in the truth table. As $P_k = A_1 \oplus A_2 \oplus \dots \oplus A_{k-1} \oplus A_k = (P_1 \oplus P_2 \oplus \dots \oplus P_{k-1}) \oplus A_k$, for two consecutive rows of the truth table, where the $k - 1$ most significant output bits are same and the A_k input bit is different, P_k output will obviously be different. So, these two output vectors will be disjoint. As all such pairs of output vectors will be disjoint, all the 2^k output vectors of the gate will be disjoint. Therefore, the gate is a reversible gate. \square

2.2 A New Family of $k*k$ Reversible Gates

It is an open problem in reversible logic to invent a gate that will realize AND, OR, and EXOR operations on the same output of the gate or on different outputs of the gate, so that by judicious selection of the operations any Boolean function can be realized by the gate. Therefore, another goal of this paper is to invent such a gate so that any Boolean function expressed in any standard form can be realized by a cascade of these gates. For this purpose, we proposed a new generalized $k*k$ reversible gate family in Figure 2(b), where f_{k-2} is an arbitrary

function of A_1, A_2, \dots, A_{k-2} and f'_{k-2} is the complement of the function f_{k-2} . The gate is a $(k-2)$ -through gate. Depending on f_{k-2} , many possible gates can be constructed. By Theorem 2, it can be shown that the gate is a reversible gate.

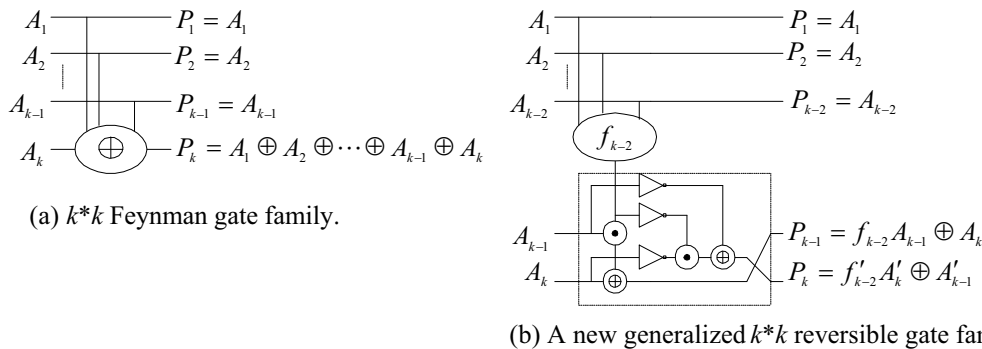


Figure 2: New families of $k*k$ reversible gates.

Theorem 2: *The $k*k$ gate described by the equations $\{P_1 = A_1, P_2 = A_2, \dots, P_{k-2} = A_{k-2}, P_{k-1} = f_{k-2}A_{k-1} \oplus A_k, P_k = f'_{k-2}A'_k \oplus A'_{k-1}\}$, where f_{k-2} is an arbitrary function of A_1, A_2, \dots, A_{k-2} and f'_{k-2} is the complement of the function f_{k-2} , is a reversible gate.*

Proof. As the gate has k inputs, the truth table of the gate will have 2^k rows. The most significant $k-2$ outputs of the gate are pass-through outputs. So, each of the 2^{k-2} possible combinations of these $k-2$ most significant output bits will appear in a group of $2^2 = 4$ in the truth table. For the group of such four consecutive rows of the truth table, where the $k-2$ most significant output bits are same, the least significant output bits P_{k-1} and P_k will depend on A_{k-1} , A_k , and f_{k-2} . Table 1 shows the possible values of P_{k-1} and P_k for $f_{k-2} = 0$ and $f_{k-2} = 1$.

Table 1: Determination of values of P_{k-1} and P_k depending on A_{k-1} , A_k , and f_{k-2} .

$A_{k-1}A_k$	$f_{k-2} = 0$	$f_{k-2} = 1$
	$P_{k-1}P_k$	$P_{k-1}P_k$
0 0	0 0	0 1
0 1	1 1	1 1
1 0	0 1	1 0
1 1	1 0	0 0

From Table 1, we see that for the group of four consecutive rows of the truth table, where the $k-2$ most significant output bits are same, the combinations of the least significant output bits P_{k-1} and P_k are disjoint. So, the group of four output vectors, corresponding to the same most significant $k-2$ output bits, will be disjoint. As all such groups of output vectors will be disjoint, all the 2^k output vectors of the gate will be disjoint. Therefore, the gate is a reversible gate. \square

Input lines A_{k-1} and A_k are used as control inputs. Depending on the values of these input signals, the gate can be used in many different modes of operations in cascades as shown in Figure 3. From Figure 3, it can be seen that NOT, AND, OR, NAND, NOR, and EXOR operations can be realized on the outputs P_{k-1} and P_k of the gate by controlling the inputs A_{k-1} and A_k .

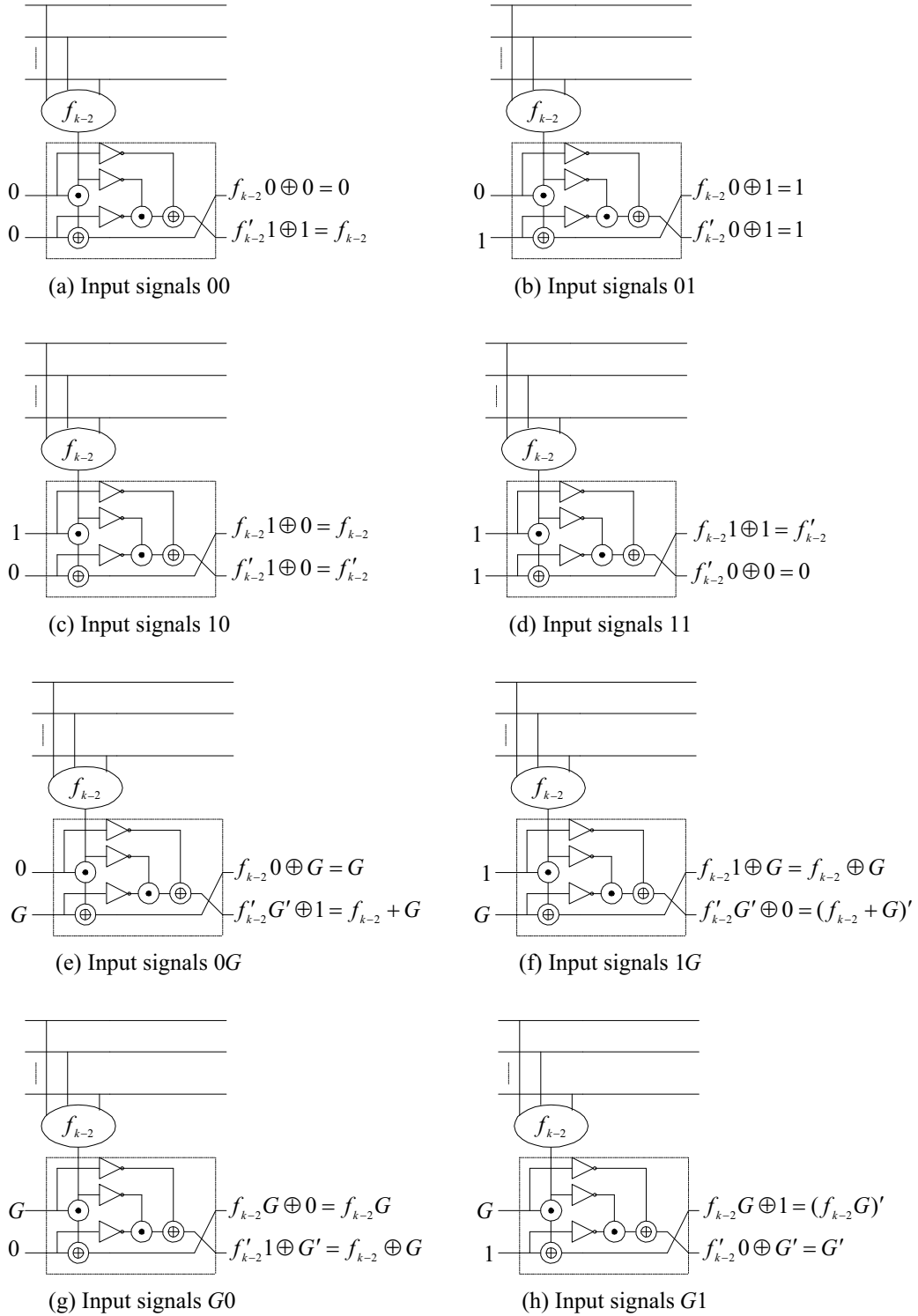


Figure 3: Different modes of operations of the new generalized $k*k$ reversible gate family that are used in cascades.

3. Synthesis of Multi-Output (Factorized) SOP

3.1 Synthesis of Multi-Output SOP

For SOP realization using cascades of new $k*k$ gates, we assume that $f_{k-2} = \prod_{i \in \{1,2,\dots,k-2\}} A_i$, that is, f_{k-2} is a product of some of the input variables A_1, A_2, \dots, A_{k-2} . For realizing a SOP, we will use the operating modes of Figure 3(a) and 3(e).

In multi-output SOP, some sub-expressions may be common among more than one SOP. Getting advantage of such a common sub-expression can not be handled by a method similar to those used in circuits with classical gates. The main constraint is that the fan-out of all signals in a reversible gate is one. Therefore, this problem has to be handled in a different way to achieve maximum benefit.

In the synthesis method for multi-output SOP using the cascades of new gate family, it is assumed that the multi-output function has been already minimized and is available in a SOP format. The method is illustrated below using the arbitrary multi-output SOP: $F_1 = BC' + AB' + A'B'C$, $F_2 = BC' + AB' + A'BC$, and $F_3 = AB' + A'BC + AC'$.

Step 1. The product sharing information among the different SOPs is summarized in Table 2. The table shows the occurrence of all product terms in different SOPs and the total number of occurrences of each product term.

Table 2: Product sharing information in multi-output SOP.

Function	F_1	F_2	F_3	No of Occurrence
BC'	X	X		2
AB'	X	X	X	3
$A'B'C$	X			1
$A'BC$		X	X	2
AC'			X	1

Step 2. Depending on the information of the product sharing table, the *connectivity tree* (or forest of trees) for the product terms is constructed as shown in Figure 4(a). In the connectivity tree each node represents a product term. The product term(s) with the highest number of occurrences is placed at the top level. Then the product terms with the decreasing numbers of occurrences are placed below. The nodes are connected by directed edges so that ORing the products in a path forms a function. For example, $AB' + BC' + A'B'C$ forms the function F_1 . A node must not have more than one inward edge and if needed a separate node is created for the product term. For example, two nodes are created for the product term $A'BC$. Though the fan-out of a signal is limited to one, multiple outward edge from a node will be allowed and this will be handled in an efficient way.

Step 3. From the connectivity tree, the *implementation graph* is created as shown in Figure 4(b). In this graph a node represents a new $k*k$ gate. The left inward edge represents A_{k-1} input line and the right inward edge represents A_k input line. The left outward edge represents P_{k-1} output line and the right outward edge represents P_k output line. The product inside the node represents the function f_{k-2} . In all nodes of the implementation graph, the A_{k-1} input is set to 0 and the A_k input is used as the signal propagation path. At the top node A_k input is also set to 0 to realize the first product term. In the connectivity tree, some nodes have multiple outward edges requiring multiple fan-out. It is clear that the left outward edge of an implementation node copies the input value at A_k and thus provides the fan-out of that value. This property is used to manage multiple fan-out requirement of the connectivity tree.

Step 4. The implementation graph is realized by cascades of new $k*k$ gates as shown in Figure 4(c). The 0 at the P_{k-1} output of the first gate is connected to the A_{k-1} input of the next gate to reduce the input constants as well as garbage outputs.

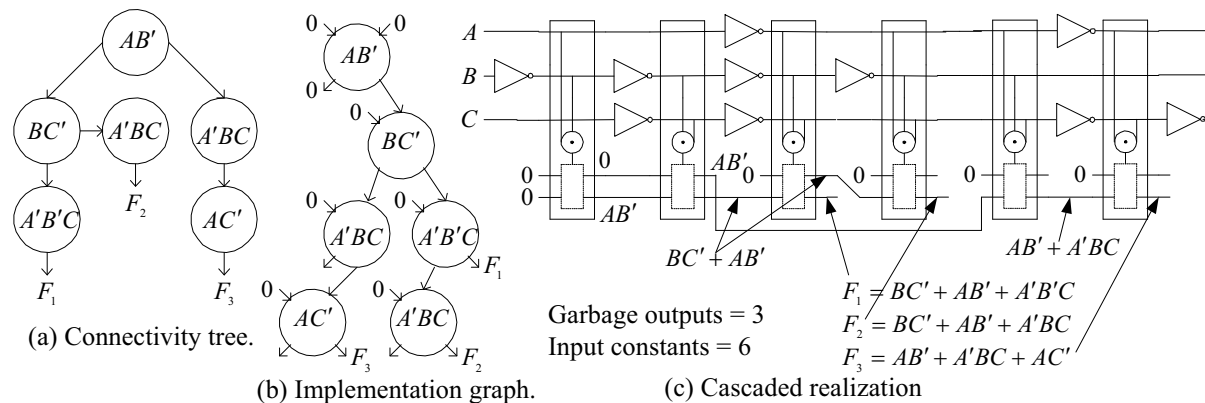


Figure 4: Synthesis of multi-output SOP.

In this particular example, 6 gates are needed, only 3 garbage outputs are generated (passing through primary input signals along the cascade is not counted as garbage output) and 6 input constants are needed. In general, the number of input constants is exactly equal to the number of gates in the cascade. However, the number of gates and the number of garbage outputs depend on the sharing of the sub-expressions in different SOPs. For a single-output SOP realization, the number of gates is equal to the number of product terms in minimized SOP. The number of garbage outputs is one less than the number of product terms and the number of input constants is exactly equal to the number of gates in the cascade.

3.2 Synthesis of Factorized SOP

For factorized SOP realization using cascades of new $k*k$ gates, we assume that f_{k-2} may be either $f_{k-2} = \prod_{\exists i \in \{1,2,\dots,k-2\}} A_i$ or $f_{k-2} = \sum_{\exists i \in \{1,2,\dots,k-2\}} A_i$, that is, f_{k-2} may be a product or sum of some of the input variables A_1, A_2, \dots, A_{k-2} . For realizing a factorized SOP, we will use the operating modes of Figures 3(a), 3(e), and 3(g).

Synthesis of factorized SOP is illustrated using the symmetric function $F_2^4 = \sum_{1 \leq i < j \leq 4} x_i x_j = x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 x_4$. This function can be factorized as $F_2^4 = (x_1 + x_2)(x_3 + x_4) + x_1 x_2 + x_3 x_4$. The implementation graph is shown in Figure 5(a). The implementation graph is realized by cascades of new $k*k$ gates as shown in Figure 5(b). In this implementation 4 gates are needed. It generates 3 garbage outputs and needs 4 input constants. In this method, it is assumed that the SOP is already factorized.

Multi-output factorized SOP can be realized using the method from Section 3.1.

4. Synthesis of EXOR of (factorized) SOPs

In many cases AND-OR-EXOR realization of a function requires fewer gates than ESOP realization [23]. Though in most of the presented minimization methods EXOR of only two SOPs are considered, in our synthesis method EXOR of any number of SOPs can be realized. Therefore, it could be a good research area to find a method for minimizing the function reduced as an EXOR of unlimited number of SOPs.

In the synthesis method for EXOR of SOPs, it is assumed that the function is already minimized as EXOR of SOPs. In this method, the SOPs are realized using the method of Section 3.1. Then the SOP outputs are EXOR-

summed using a $k*k$ Feynman gate. Realization of an arbitrary EXOR of three SOPs (the SOPs from example of Figure 4) is shown in Figure 6. In a similar method, EXOR of factorized SOPs can be realized.

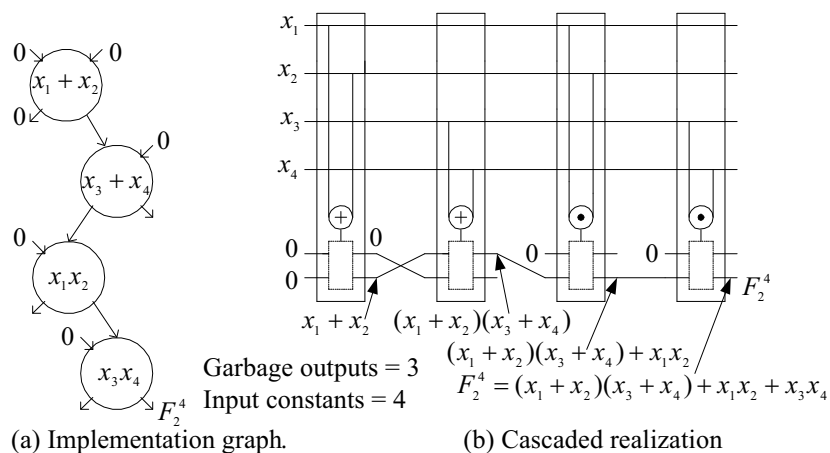


Figure 5: Synthesis of single-output factorized SOP.

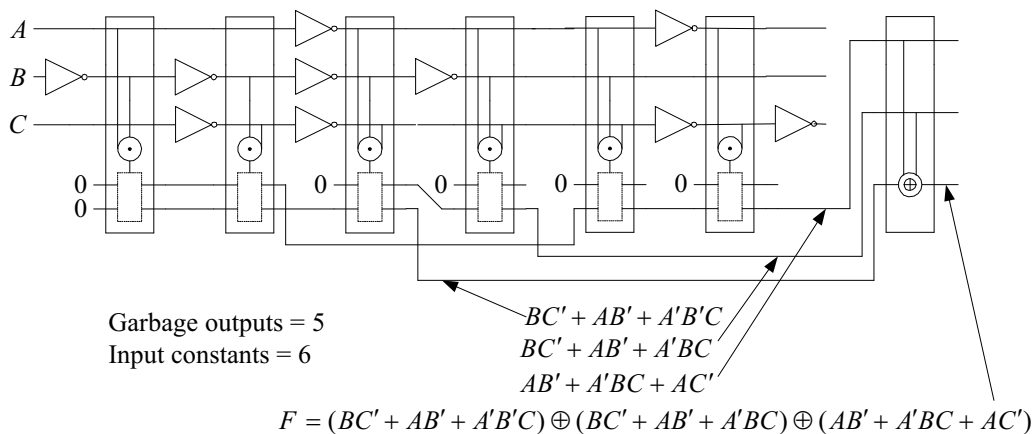


Figure 6: Synthesis of EXOR of SOPs.

5. Synthesis of Single-Output (factorized) ESOP

5.1 Synthesis of Single-Output ESOP

For single-output ESOP realization we use the operating modes of Figures 3(a) and 3(g). The synthesis method for single-output ESOP using the new $k*k$ gate is illustrated using the arbitrary ESOP: $F = BC' \oplus AB' \oplus A'B'C$. In this method, it is assumed that the function is already minimized as ESOP. The implementation graph and the cascaded realization are shown in Figure 7(a) and 7(b), respectively. In the implementation graph, the A_k inputs of all the nodes are set to 0 and the A_{k-1} inputs are used as signal propagation path. The A_{k-1} input of the first node is also set to 0 to produce the first product term. Some of the P_{k-1} outputs may generate 0s as they produce product of two terms. These 0s are identified and indicated in the graph. Moreover, the ordering of the products in the graph can be rearranged to increase the chance of obtaining such 0 output. Each such 0 is connected to the A_k input of the next gate of the cascade to reduce the garbage outputs as well as input constants. In this particular example, three gates are needed. The circuit has one garbage output and needs 2 input constants.

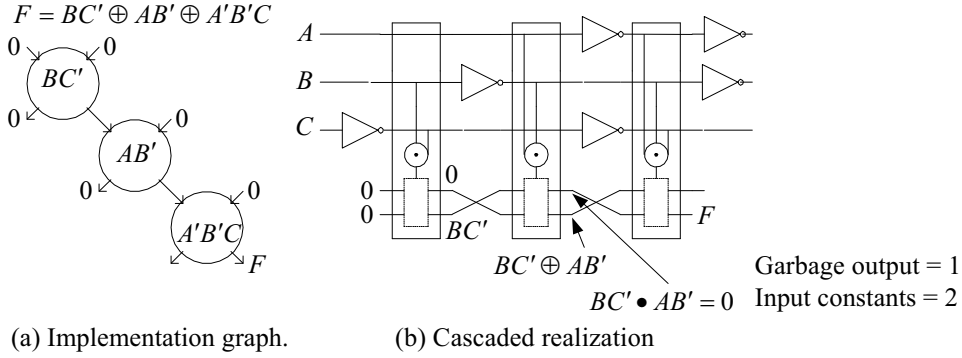


Figure 7: Synthesis of single-output ESOP.

5.2 Synthesis of Single-Output Factorized ESOP

For factorized ESOP realization using cascades of new $k \times k$ gates, we assume that f_{k-2} may be either $f_{k-2} = \prod_{i \in \{1, 2, \dots, k-2\}} A_i$ or $f_{k-2} = \sum_{i \in \{1, 2, \dots, k-2\}} \oplus A_i$, that is, f_{k-2} may be a product or EXOR-sum of some of the input variables A_1, A_2, \dots, A_{k-2} . For realizing a factorized ESOP, we will use the operating modes of Figures 3(a) and 3(g).

Synthesis of single-output factorized ESOP is illustrated using the symmetric function $E_2^4 = \sum_{1 \leq i < j \leq 4} \oplus x_i x_j = x_1 x_2 \oplus x_1 x_3 \oplus x_1 x_4 \oplus x_2 x_3 \oplus x_2 x_4 \oplus x_3 x_4$. This function can be factorized as $E_2^4 = (x_1 \oplus x_2)(x_3 \oplus x_4) \oplus x_1 x_2 \oplus x_3 x_4$. The implementation graph and the cascaded realization are shown in Figure 8(a) and 8(b), respectively. In this implementation 4 gates are needed. It generates 3 garbage outputs and needs 4 input constants. In this method, it is assumed that the ESOP is already factorized.

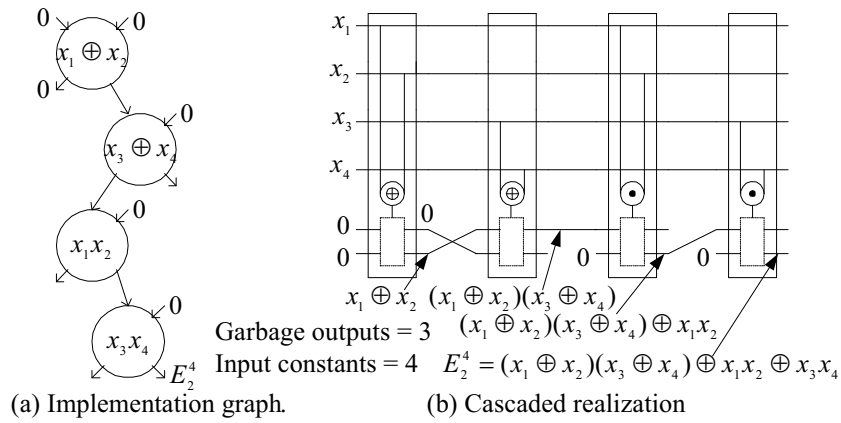


Figure 8: Synthesis of single-output factorized ESOP.

6. Design of Full-Adder Circuit

The full-adder circuit is described by the functions $S = A \oplus B \oplus C_i$ and $C_o = (A \oplus B)C_i \oplus AB$. The implementation graph and the cascaded realization of the full-adder circuit are shown in Figure 9(a) and 9(b), respectively. In this full-adder implementation three gates are needed. It generates 1 garbage output (passing through primary input signals are not counted as garbage output) and needs 2 input constants.

The full-adder implementation of Khlopotine, Perkowski, and Kerntopf [18] and Perkowski et al [21] requires two Toffoli gates and two Feynman gates with no garbage output (pass through primary inputs are not counted

as garbage output) and only one input constant. Therefore, our full-adder realization is more efficient than the realization of [18] and [21] as it requires one less gate with increase of one garbage output and one input constant. The full-adder design of Miller and Dueck [25] requires 5 generalized Toffoli gates. So, again our full-adder design is more efficient than the design of [25] in terms of gate count.

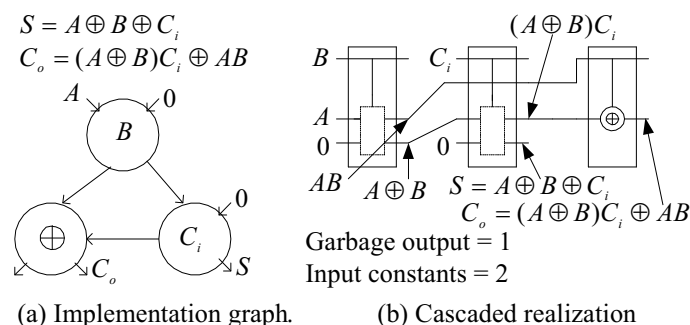


Figure 9: Design of full-adder circuit.

A *Pseudoproduct* is a product of EXOR factors. For example, $(x_1 \oplus x_2') \cdot x_5 \cdot (x_1 \oplus x_4 \oplus x_7')$ is a Pseudoproduct. From the single-output ESOP and full-adder examples, it can be guessed that EXOR of Pseudoproducts may yield good results with the proposed cascade. Though minimization of OR of Pseudoproducts expression has been recently studied [24], no work has yet been done on minimization of EXOR of Pseudoproducts expression. This could be a good new research area.

7. Experimental Results

A program for synthesizing multi-output SOP cascades using the method of Section 3.1 has been developed in C language and named REV-SOP. In REV-SOP program, it is assumed that the multi-output SOP is already minimized. Experimentation with 36 MCNC benchmark functions has been done and the results are given in Table 3. In column 5, the number of gates in the multi-output SOP cascade is shown. The number of garbage outputs generated in the cascade is shown in column 6 and the input constants (0s) needed for the cascade is shown in column 7. In the classical multi-output SOP realization more than one fan-out of a gate is permissible, but in the reversible circuit fan-out of all signals are restricted to only one. For this reason, in the connectivity tree, the inward edge of a node is restricted to one and if needed additional nodes are created for a product term (see Figure 4(a)). Each of these nodes represents a gate in the resultant cascade. Therefore, it is clear that the number of gates in the cascade will be at least equal to the number of product terms in the multi-output SOP and in many cases this number will be greater than the number of product terms. The ratio of the number of gates in the cascade to the number of products in the multi-output SOP is shown in column 8. This value can be interpreted as the average number of nodes created for a product term. For example, in 5xp1 function on average 1.292 nodes are created for a product term. In the multi-output SOP cascade, P_{k-1} outputs of some gates are left unused as garbage outputs. The ratio of the garbage outputs to the number of gates in the cascade is shown in the last column. This value can be interpreted as the average number of garbage outputs created per gate. For example, in 5xp1 function on average a gate produces 0.881 garbage output. Alternatively, it can be said that 88.1% gates produce garbage output.

For four single-output SOP functions (9sym, 9symml, t481, and xor5), the number of gates in the cascade is exactly equal to the number of products in the SOP. For 10 multi-output SOP functions, the number of gates in the cascade is also exactly equal to the number of products in the SOP. For 13 multi-output SOP functions, the number of gates in the cascade is almost equal to the number of products in the SOP. ***These experimental results show that the sub-expression sharing technique of the developed algorithm is so efficient that the number of reversible gates (with single fan-out capability) in the resultant multi-output SOP cascade is almost equal to the number of products (classical AND gates with multiple fan-out capability) in the multi-output SOP.*** For the remaining nine functions, the number of gates in the cascade is double or higher times than

the number of products in the SOP. The nature of product sharing among the outputs of these functions is such that in the connectivity tree many nodes are needed to be created for a product term.

Table 2. Experimental results of multi-output SOP cascades of new reversible gates for some MCNC benchmark functions.

MCNC Benchmark Functions (expressed as SOP)				Multi-output SOP Cascades				
Function Name	No. of Inputs	No. of Outputs	No. of Products	No. of Gates	No. of Garbage Outputs*	No. of Input Constants	No. of Gates in Cascade : No. of Products in SOP	No. of Garbage Output : No. of Gates
5xp1	7	10	65	84	74	84	1.292	0.881
9sym	9	1	86	86	85	86	1.000	0.988
9symml	9	1	87	87	86	87	1.000	0.989
adr2	4	3	11	11	8	11	1.000	0.727
alu2	10	6	141	141	135	141	1.000	0.957
alu4	14	8	577	589	581	589	1.021	0.986
b12	15	9	43	51	42	51	1.186	0.822
bw	5	28	22	225	197	225	10.227	0.876
c8	28	18	79	79	61	79	1.000	0.772
cc	21	20	45	45	25	45	1.000	0.556
clif	9	5	120	134	129	134	1.117	0.963
con1	7	2	9	9	7	9	1.000	0.778
cu	14	11	19	22	11	22	1.158	0.500
duke2	22	29	87	220	191	220	2.529	0.868
ex5	8	63	74	1427	1364	1427	19.284	0.956
f51m	8	8	76	76	68	76	1.000	0.895
frg1	28	3	119	119	116	119	1.000	0.975
il	25	16	28	38	22	38	1.357	0.579
inc	7	9	30	56	47	56	1.867	0.839
misex1	8	7	12	41	34	41	3.417	0.829
misex2	25	18	28	29	11	29	1.036	0.379
misex3c	14	14	196	248	234	248	1.265	0.944
pdc	16	40	142	411	371	411	2.894	0.903
rd53	5	3	31	31	28	31	1.000	0.903
rd73	7	3	127	127	124	127	1.000	0.976
rd84	8	4	255	259	255	259	1.016	0.985
sao2	10	4	58	76	72	76	1.310	0.947
seq	41	35	336	1647	1612	1647	4.902	0.979
spla	16	46	249	610	564	610	2.450	0.925
squar5	5	8	25	29	21	29	1.160	0.724
t481	16	1	481	481	480	481	1.000	0.998
table3	14	14	176	596	582	596	3.386	0.977
table5	17	15	158	571	556	571	3.614	0.974
vg2	25	8	110	110	102	110	1.000	0.927
xor5	5	1	16	16	15	16	1.000	0.938
Z5xpl	7	10	65	111	101	111	1.708	0.910

* Passing-through primary input signals along the cascade are not counted as garbage output.

For single-output SOP, the number of garbage outputs (passing-through primary input signals along the cascade are not counted as garbage outputs) generated in the cascade is one less than the number of reversible gates in the cascade and this is reflected in the result of the single-output SOP functions. For multi-output SOP, the

number of garbage outputs generated in the cascades depends on the nature of sub-expression sharing among the outputs. The experimental results show that this value is relatively high. However, the number of garbage signals can be minimized by the use of local mirror and spy circuits [12, 21]. This would be a topic for further research.

In the new $k*k$ gates, constant 0s are applied to the control inputs for selecting the mode of operation of the gate in the cascade (see Figure 3). For both single-output and multi-output SOPs, the number of input constants is exactly equal to the number of reversible gates in the cascade. This number is high and needs to be reduced in the future research.

8. Conclusion

Contribution of This Paper

Some generalizations of the well-known Feynman gate are known from literature [18, 22]. *We proposed a new $k*k$ generalization of the Feynman gate. We also proposed a new $k*k$ reversible gate family, which shows very good performance in realizing multi-output SOP cascades.*

To our knowledge, no method has been yet reported to synthesize multi-output (factorized) SOP and EXOR of (factorized) SOPs using reversible gates. In this paper, for the first time, *we presented a synthesis method for multi-output (factorized) SOP using cascades of the new $k*k$ gate family.* For utilizing the benefit of sub-expression sharing among the outputs of the multi-output SOPs, *two graph-based data structures* – connectivity tree and implementation graph *are introduced.* For the first time, *we also presented a synthesis method for EXOR of (factorized) SOPs using cascades of the new $k*k$ gates and a new generalized $k*k$ Feynman gate, where the fan-in of the EXOR gate is not limited.* Another synthesis method for single-output (factorized) ESOP function using the new $k*k$ gates is also presented, where judicious rearrangement of the products in the implementation graph reduces the number of garbage outputs and input constants.

We also presented design of full-adder circuit with only two new gates and one Feynman gate, which is more efficient than the designs of [18], [21], and [25] in terms of gate count.

We wrote a C program named REV-SOP to synthesize multi-output SOP cascades and experiments were done with 36 MCNC benchmark functions. *The experimental results show that the sub-expression sharing technique of the developed algorithm is so efficient that the number of reversible gates (with single fan-out capability) in the resultant multi-output SOP cascade is almost equal to the number of products (classical AND gates with multiple fan-out capability) in the multi-output SOP.*

All the synthesis methods presented in this paper are technology independent and can be used in association with any known or future reversible technology. Observe that we use complex gates that are programmable, since function f_{k-2} is an arbitrary Boolean function. The proposed method will bring the best results for those technologies where complex (programmable) gates can be built but the connectivity patterns are constrained because of the attempt to reduce connection cost, which dominates large circuits. It is believed by many specialists that such structures will be of primary importance in several nano-technologies. Thus the circuit's regularity and short connections, as in cascades, will be advantageous. Finally, as proposed in [26], complex gates can be macro-generated to structures of simpler gates that are good for any given technology. Next, the local backtracking transformations, particular for this technology, can be used to optimize the circuit. Thus, if proposing such complex gates as those introduced here is practical, can be verified only experimentally for any particular reversible realization technology.

Further Works

The future research includes: (i) investigating the possibility of reducing the number of garbage outputs in the multi-output SOP cascades by the use of local mirror and spy circuits or other techniques, (ii) investigating the way of reducing the input constants needed for the multi-output SOP cascades, (iii) writing C program to synthesize multi-output factorized SOP cascades, (iv) developing minimization method for EXOR of any

number of (factorized) SOPs, (v) developing synthesis method for multi-output (factorized) ESOP function using cascades of generalized $k*k$ reversible gates, and (vi) developing minimization method for EXOR of Pseudoproducts.

References

- [1] R. Landauer, Irreversibility and heat generation in the computational process. *IBM Journal of Research and Development*, 5, pp. 183-191, 1961.
- [2] J. Birnbaum. Computing alternatives. *Talk given at ACM97*, March 3, 1997, San Jose, California.
- [3] C. Bennett, Logical reversibility of computation. *IBM Journal of Research and Development*, 17, pp. 525-532, 1973.
- [4] D. Deutsch, Quantum computational networks. *Proc. Roy. Soc. Lond. A* 425, 1989, pp. 73 – 90.
- [5] A. De Vos, Towards reversible digital computers. *Proceedings of European Conference on Circuit Theory and Design*, Budapest, pp. 923-931, 1997.
- [6] B. Desoete, A. De Vos, M. Sibinski, and T. Widerski, Feynman's reversible logic gates implemented in silicon. *Proceedings of 6th International Conference MIXDES*, pp. 496-502, 1999.
- [7] P. Picton, Optoelectronic, multivalued, conservative logic. *International Journal of Optical Computing*, 2, pp. 19-29, 1991.
- [8] P. Picton, A universal architecture for multiple-valued reversible logic. *MVL Journal*, 5, pp. 27-37, 2000.
- [9] J. A. Smolin, and D. P. DiVincenzo, Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate. *Physical Review A*, 53, 1996, pp. 2855-2856.
- [10] A. Peres, Reversible logic and quantum computers, *Physical Review A*, 32, pp. 3266-3276, 1985.
- [11] R. C. Merkle, Two types of mechanical reversible logic. *Nanotechnology*, 4, pp. 114-131, 1993.
- [12] E. Fredkin and T. Toffoli, Conservative logic. *International Journal of Theoretical Physics*, 21, pp. 219-253, 1982.
- [13] P. Kerntopf, A comparison of logical efficiency of reversible and conventional gates. *Proceedings of 9th IEEE Workshop on Logic Synthesis*, pp. 261-269, 2000.
- [14] P. Kerntopf, Maximally efficient binary and multi-valued reversible gates. *Proceedings of ULSI Workshop*, Warsaw, Poland, May 2001, pp. 55 – 58.
- [15] L. Storme, A. De Vos, and G. Jacobs, Group theoretical aspects of reversible logic gates. *Journal of Universal Computer Science*, 5, pp. 307-321, 1999.
- [16] P. Kerntopf, Synthesis of multipurpose reversible logic gates. *Proceedings of EUROMICRO Symposium on Digital Systems Design*, 2002, pp. 259 – 266.
- [17] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, A. Coppola, B. Massey, Regularity and symmetry as a base for efficient realization of reversible logic circuits. *Proceedings of IWLS 2001*, pp. 90-95, 2001.
- [18] A. Khlopov, M. Perkowski, and P. Kerntopf, Reversible logic synthesis by gate composition. *Proceedings of IWLS 2002*. pp. 261 – 266.
- [19] M. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Mishchenko, M. Chrzanowska-Jeske, Three-dimensional realization of multi-valued functions using reversible logic. *Proceedings of ULSI 2001 Workshop*, Warsaw, Poland, May 2001. pp 47 – 53.
- [20] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, A. Coppola, B. Massey, Regular realization of symmetric functions using reversible logic. *Proceedings of Euro-Micro 2001*, pp. 245 – 252.
- [21] M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al-Rabadi, A. Coppola, A. Buller, X. Song, M. M. H. A. Khan, S. Yanushkevich, V. Shmerko, and M. Chrzanowska-Jeske, A general decomposition for reversible logic. *Proceedings of RM 2001*. pp. 119 – 138.
- [22] A. Mishchenko and M. Perkowski, Reversible Maitra cascades for single-output functions. *Proceedings of IWLS 2002*. pp. 197 – 202.
- [23] D. Debnath and T. Sasao, A heuristic algorithm to design AND-OR-EXOR three-level networks. *Proceedings of ASP-DAC'98*, Yokohama, Japan, 1998, pp. 69-74.
- [24] V. Ciriani and A. Bernasconi, 2-SPP: a practical trade-off between SP and SPP synthesis. *Proceedings of IWBP 2002*. pp. 133 –140.
- [25] D. M. Miller and G. W. Dueck, Spectral techniques for reversible logic synthesis. *Proc. RM 2003*.
- [26] M. Perkowski et al, A Hierarchical approach to computer-aided design of quantum circuits, *Proc. RM 2003*.