Portland State University

PDXScholar

Electrical and Computer Engineering Faculty Publications and Presentations

Electrical and Computer Engineering

6-2002

Logic Synthesis of Reversible Wave Cascades

Alan Mishchenko
Portland State University

Marek Perkowski
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/ece_fac

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Citation Details

Alan Mishchenko, Marek Perkowski, "Logic Synthesis for Reversible Wave Cascades", International Workshop on Logic and Synthesis, New Orleans, Louisiana, June 2002.

This Conference Proceeding is brought to you for free and open access. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications and Presentations by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Logic Synthesis of Reversible Wave Cascades

Portland Quantum Logic Group

Alan Mishchenko and Marek Perkowski

Department of Electrical and Computer Engineering
Portland State University
Portland, OR 97207, USA
[alanmi, mperkows]@ece.pdx.edu

Abstract

A circuit is reversible if it maps each input vector into a unique output vector, and vice versa. Reversible circuits lead to power-efficient CMOS implementations. Reversible logic synthesis may be applicable to optical and quantum computing. Minimizing garbage bits is the main challenge in reversible logic synthesis.

This paper introduces an algorithm to generate the cascade of reversible complex Maitra terms (called here reversible wave cascade) implementing incompletely specified Boolean functions. The remarkable property of the presented method compared to other reversible synthesis methods is that it creates at most one constant input and no additional garbage outputs. Preliminary estimation suggests that the method may be applicable to small and medium-sized benchmarks.

1 Introduction

Reducing power becomes the main task of modern digital circuit design. One may ask, what are the limits of low-power design? The answer was given in the paper of Bennett and Landauer [2], who proved that losing information is equivalent to losing power. An example of a circuit that loses information is a two-input AND gate, which produces value 0 for the three combinations of input values: 00, 01 and 10. Thus, the values of inputs cannot be determined from the value of the output.

The gate that does not lose information is called *reversible*. For instance, the Feynman gate described by the equations $\{P = A, Q = A \oplus B\}$ is reversible, because for each combination of signals $\{P, Q\}$, there is exactly one combination of signals $\{A, B\}$.

Logic circuits consume energy because of technological factors (such as power dissipation while switching) and because of the loss of information. While the first component is constantly decreasing due to the improvement of the implementation technologies and the emergence of new design principles, such as adiabatic design, the second part of the energy consumption is related to information and can be decreased (to zero) only by adopting the reversible design principles.

As of year 2002, the second component of energy consumption is much smaller but if the progress in low-power technology follows Moore's law, the second component will start dominating around year 2020. According to [2], it is a necessary condition to use only reversible gates to build a logic circuit that does not consume energy[1].

It was shown that reversible gates can be built in CMOS [5][6], DNA [19], optical and other technologies, and that all quantum logic gates are reversible [20]. A challenging goal might be to develop a system to synthesize reversible implementations of Boolean functions and state machines.

The difference of reversible logic synthesis compared to binary logic synthesis can be summarized as follows:

1. The gates used to implement the circuit have the equal number of inputs and outputs.
2. Every output of a gate, which is not used in the circuit, is a *garbage signal*. A good synthesis method minimizes the number of garbage signals.
3. The total number of constants at inputs of the gates is kept as low as possible.
4. A gate output can be used only once (the fanout count of each output is equal to one). If two copies of a signal are required, a copying circuit is used.
5. The resulting circuit is acyclic.

In addition to the Feynman gate mentioned above, the literature discusses Toffoli [23] and Fredkin [7] gates and their construction using existing and future technologies. Three-input three-output gate families have been analyzed

[1] Energy may be lost for input and output operations.

in [8][2]. As a result of this analysis, several new types of binary and multiple-valued reversible gates have been created.

In another line of research, the concepts of regular structures, such as PLAs [19], 2-dimensional lattices [15][16], three-dimensional lattices and nets were adapted to reversible logic. The methods based on decision diagrams have been proposed, as well as composition and decomposition methods [16][9]. Most of the reversible gates in literature are three-input three-output or four-input four-output gates, except for papers [5][6][20], in which restricted multi-input, multi-output gates are presented without systematic design methods.

In this paper, we introduce k-input k-output reversible gates for $k > 4$. To our knowledge, no systematic methods for synthesis using gates with $k > 3$ have been published. The proposed synthesis method satisfies all of the requirements for reversible logic synthesis. The algorithm heuristically minimizes the number of Maitra terms [13], and therefore reduces the delay of the resulting circuit. The method is based on Boolean properties of functions and employs satisfiability implemented using Binary Decision Diagrams [3].

The rest of the paper is organized as follows. Section 2 gives the overview of Toffoli family of gates. Section 3 describes the structure of the reversible wave cascade constructed using Toffoli family of gates. Section 4 introduces the mathematical background to discuss logic synthesis of wave cascade. Section 5 presents the logic synthesis algorithm. Section 6 shows preliminary experimental results. Section 7 concludes the paper.

2 Toffoli Family of Gates

Feynman gate is described by equations:

$$P = A,\ Q = A \oplus B.$$

Toffoli gate [23] is described by equations:

$$P = A,\ Q = B,\ R = AB \oplus C.$$

Feynman gate can be generalized to the following family of gates called 1*1 *family of Toffoli gates*:

$$P = A,\ Q = f_1(A) \oplus B,$$

where f_1 is an arbitrary function of one variable. There are only four functions of one variable in binary logic.

Similarly, Toffoli gate can be generalized to the family called 2*2 *family of Toffoli gates*:

$$P = A,\ Q = B,\ R = f_2(A, B) \oplus C,$$

where f_2 is an arbitrary function of two variables.

[2] Such gates are the minimal universal gates in binary reversible logic, but interestingly, two-input two-output gates are the minimal universal gates in quantum logic [20].

Next, the concept of Toffoli gate can be generalized to a Toffoli family with an arbitrary number of inputs called *k***k family of Toffoli gates*:

$$P_1 = A_1,\ P_2 = A_2,\ \ldots,\ P_{n-1} = A_{n-1},$$
$$P_n = f_{n-1}(A_1, A_2, \ldots, A_{n-1}) \oplus A_n,$$

where f_{n-1} is an arbitrary function of n-1 variables.

It is easy to prove using the definition of reversible logic that all gates in the 1*1, 2*2, and k*k families of Toffoli gates are reversible.

In particular, functions f_i can be arbitrary conjunctions of Boolean variables and, therefore, the cascade of k*k gates can realize the circuit described by Positive Polarity Reed-Muller Form (PPRM). This cascade has as many gates as there are terms in PPRM and as many horizontal lines as there are input variables plus one (see Figure 1).

The additional input is an input to the first EXOR gate in the cascade. It can be set to constant 0 or constant 1 Boolean function[3]. Thus, an arbitrary k-input single-output function can be realized by a reversible circuit constructed from the gates of the (k+1)*(k+1) family of Toffoli gates, in which functions f_{n-1} are conjunctions of some of the input variables, $A_1, A_2, \ldots, A_{n-1}$.

In Fixed Polarity Reed-Muller Form (FPRM), every input variable can be negated (negative variable) or not negated (positive variable) but cannot be both positive and negative at the same time. The cascade can also realize FPRM. This is done using inverters[4] for the inputs corresponding to the negated input variables.

Finally, ESOP can be realized in two ways:

(1) by inserting inverters into the input lines if the given term has a negated variable,

(2) by building reversible gates with internal inverters[5].

In this paper, we present the circuit structure called *reversible wave cascade* based on k*k family of Toffoli gates, in which f_{n-1} are arbitrary two-variable functions. In CMOS, the complex k*k gates can be built directly from transistors, which saves constant inputs and outputs.

3 Structure of Wave Cascade

The definitions presented in this section are based on [12] and [13], with some modifications.

Definition. A *complex Maitra term* is recursively defined as follows:

(1) Constant 0 (1) Boolean function is a Maitra term.

(2) A literal is a Maitra term.

(3) If M_i is a Maitra term, a is a literal, and G is an arbitrary two-input Boolean function, then $M_{i+1} = G(\ a, M_{i+1}\ )$ is a Maitra term.

[3] Constant 1 corresponds to the first term of PPRM.

[4] Inverter is a reversible gate.

[5] The second implementation is likely to be faster.

Additionally, it is required that each variable appears in each Maitra term only once and that the same variable ordering is used to represent all Maitra terms.

Previous authors [10][11][12] restricted the two-input functions used in the Maitra terms to only functions AND, OR, and EXOR. For them, it was important to have a small number of logic functions, because they considered the Maitra term as a basic structure to build programmable logic devices. For the purposes of reversible logic synthesis, on the other hand, it is better to use the above more general definition.

In a variation of our algorithm targeting low-power CMOS implementation, we do impose a restriction on the type of functions G motivated by the technological consideration. In this case, G cannot be an EXOR function and its complement, NEXOR. The mathematical formulation of the problem introduced in the following sections accommodates this restriction.

Definition. The *reversible wave cascade* for a completely specified Boolean function F is a set of Maitra terms, M_i, $1 \le i \le n$, such that $F = \sum_{i=1}^{n} \oplus M_i$.

Definition. The *reversible wave cascade* for an incompletely specified Boolean function F is the cascade implementation of a completely specified Boolean function C belonging to function interval $F = (Q, R)$.

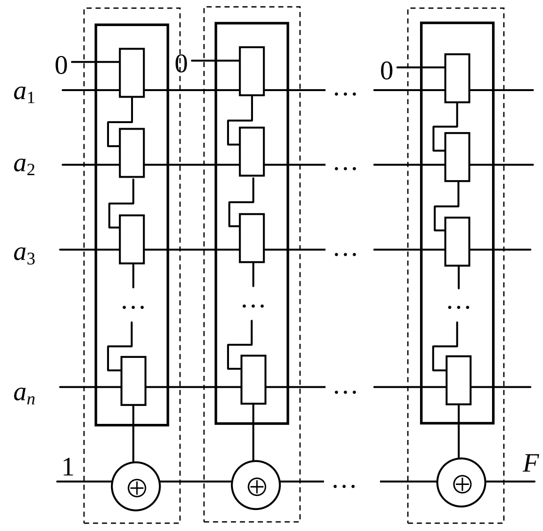

Figure 1. Reversible Wave Cascade.

The general structure of the Wave Cascade is shown in Figure 1. The inputs variables (a_1, a_2, …, a_n) are the primary inputs of function F. In the direct computation flow, they propagate from left to right and feed the two-input gates that form the individual stages of the cascade. It is assumed, without the loss of expressive power of the cascade, that one of the inputs of the topmost gate is the constant 0 Boolean function.

The outputs of the Maitra terms feed the inputs of the EXOR gates at the bottom. The EXOR gates form the cascade producing the output of function F. Without the loss of expressive power of the cascade, the input of the first EXOR gate is set to the constant 1 Boolean function.

The constant-1 input of the cascade is the only garbage input in the reversible representation of the cascade. In this implementation, the individual cascades enclosed in the dashed lines can be viewed as n+1 reversible gates belonging to $(k+1)*(k+1)$ family of Toffoli gates.

Logic synthesis of the cascades starts with the incompletely specified function $F = (Q, R)$ and returns the set of gates implementing the stages. The logic synthesis methods developed in this paper use the mathematical concepts summarized below.

4 Mathematical Background

This introduction assumes the familiarity of the reader with Boolean algebra and Binary Decision Diagrams [3].

4.1 Fundamentals

Definition. A completely specified Boolean function of n variables is a mapping of $B^n \to B$, where $B \in \{0,1\}$. An incompletely specified Boolean function F is a function interval (Q, R), where Q and R are completely specified Boolean functions, known as the *on-set* (lower bound) and the *off-set* (complemented upper bound).

The incompletely specified function is *well-defined* if the on-set and the off-set do not overlap: $Q \wedge R \equiv 0$.

Definition. A completely specified function G *belongs* to the interval $F = (Q, R)$ iff $G \wedge R \equiv 0$ and $\overline{G} \wedge Q \equiv 0$.

Definition. The *support* of a completely specified function is the set of variables, on which the function depends. The *support* of the incompletely specified function $F = (Q, R)$ is the union of supports of Q and R.

Definition. The *existential quantification* of a completely specified function $F(a)$ w.r.t. variable a_0 is the completely specified function

$$\exists_{a0}F(a) = F_{a0=0}(a) \vee F_{a0=1}(a).$$

This function can depend on all variables in a, except a_0.

Similarly, the *universal quantification* is the function $\forall_{a0}F(a) = F_{a0=0}(a) \wedge F_{a0=1}(a)$.

The following property holds: $\exists_{a0}F(a) = \overline{\forall_{a0}\overline{F}(a)}$.

Definition. The result of *removing* a_0 from the support of an incompletely specified function $F(a) = (Q(a), R(a))$ is the set of completely specified functions $G(a)$, which belong to the interval $F(a)$ and do not depend on a_0. If this set is not empty, variable a_0 *can be removed* from the support of $F(a)$.

Theorem 1. Variable a_0 can be removed from the support of $F(a) = (Q(a), R(a))$ iff

$$\exists_{a0}Q(a) \,\&\, \exists_{a0}R(a) \equiv 0.$$

The result of removing variable a_0 from the support of the incompletely specified function $F(a)$ is another incompletely specified function: $F'(a) = (\exists_{a0}Q, \exists_{a0}R)$.

Theorem 1 requires that the resulting incompletely specified function was well-defined.

Example. An incompletely specified function $F = (Q, R)$ in shown in Figure 2. This function contains four completely specified functions derived by four different assignment of don't-care minterms (101) and (010). In particular, when the value of the function is 0 in both minterms, the corresponding completely specified function is the same as Q. This function does not depend on variable a. Therefore, variable a can be removed from the support of F.

F

bc\a	0	1
00	0	0
01	0	-
11	1	1
10	-	0

Q

bc\a	0	1
00	0	0
01	0	0
11	1	1
10	0	0

R

bc\a	0	1
00	1	1
01	1	0
11	0	0
10	0	1

Figure 2. Example of incompletely specified function.

4.2 Sets of Functions

Definition. An *encoded representation* of a set of completely specified functions, which depend on the set of variables a, is a completely specified function $\varphi(a,z)$ such that substituting the assignment z_0 of variables z into $\varphi(a,z)$ yields function $\varphi_{z0}(a)$, which belongs to the set.

If the set consists of n functions, a unique encoding representation can be constructed with $k \geq \lceil \log_2 \mu \rceil$ variables z.

Example. The set of functions $\{ a_1a_2,\ a_1\bar{a}_2,\ \bar{a}_1 \}$ can be encoded using codes $\{ \bar{z}_1z_2,\ z_1\bar{z}_2, z_1z_2 \}$, which leads to the following encoded representation:

$$\varphi(a,z) = a_1a_2\ \bar{z}_1z_2 \ \vee\ a_1\bar{a}_2\ z_1\bar{z}_2 \ \vee\ \bar{a}_1\ z_1z_2 .$$

4.3 Encoded Representation of Maitra Terms

All implementations of one stage of the wave cascade, shown Figure 1, are represented using the primary input variables a and the additional variables z encoding individual gates of the cascade.

The functionality of each two-input gate is represented using four encoding variables (one variable per minterm). However, this representation is wasteful because the simultaneous complementation of the gate's input and the previous gate's output leads to an equivalent cascade.

Therefore, without limiting the expressive power of the representation, we set the value of all two-input gates in minterm (11) to be 0. We use the set of variables z with double indices. The first index stands for the number of the gate. The second stands for the number of the corresponding minterm in the gate's Karnaugh map.

The functional representation of all implementation of one stage of wave cascade with n inputs is derived recursively, assuming that the representation of the stage with n-1 inputs is known:

$$\varphi_0(a,z) = 0;$$

$$\varphi_k(a,z) = z_{k0}\,\bar{a}_k\ \overline{\varphi_{k-1}(a,z)} \ \vee z_{k1}\,\bar{a}_k\ \varphi_{k-1}(a,z) \ \vee$$

$$z_{k2}a_k\,\overline{\varphi_{k-1}(a,z)},\ \ 1 \leq k \leq n;$$

It is important that the size of the BDD representing $\varphi_n(a,z)$ is linear in the number of gates if variables a and z_{ki} are interleaved in the variable ordering.

Table 1. The number of BDD nodes and minterms in the encoding of all implementations of one stage of wave cascade.

n	$N(n)$	$M(n)$
2	9	16
3	14	52
4	19	408
5	24	3,280
6	29	26,208
7	34	209,728
8	39	1,677,696
9	44	13,421,824
10	49	107,374,080

Table 1 gives the number of nodes, $N(n)$, including the terminal node, in the complement-edge BDD of $\varphi_n(a,z)$ assuming that variables are ordered as follows: $(a_1, z_{10}, z_{11}, z_{12},\ \ a_2, z_{20}, z_{21}, z_{22}, \ldots,\ \ a_n, z_{n0}, z_{n1}, z_{n2})$. The table shows that the number of nodes is linear in the number of variables: $N(n) = 5n - 1$. $M(n)$ is the number of on-set minterms in $\varphi_n(a,z)$ computed using the BDD minterm counting procedure.

In the cascade synthesis, we use constraints representing several stages of the cascade connected by EXOR gates. The size of this representation is not linear in the number of input variables but it is manageable for small number of stages and functions of less than 10 variables.

5 Wave Cascade Synthesis Algorithm

In this section, it is assumed that the incompletely specified function to be implemented as the wave cascade is $F(a) = (Q(a), R(a))$. The set of encoded completely specified functions representing one or more stages of the cascade is $\varphi(a,z)$.

Definition. The *remainder* is the set of incompletely specified functions resulting from implementing the remaining part of the cascade, assuming that the first stages of the cascade are represented by $\varphi(a,z)$.

Using the linearity of the EXOR operation, the remainder is computed as follows:

$$F^R(a,z) = (Q^R(a,z), R^R(a,z)) = (Q(a)\oplus\varphi(a,z), R(a)\oplus\varphi(a,z)).$$

The algorithm for wave cascade synthesis is iterative. It generates one or more stages of the cascades at a time, in such a way that when the cascades are added to the implementation, the support of the remainder is reduced by at least one variable.

The following theorem allows us to select the representatives of the encoding set of functions, which do not depend on some input variables.

Theorem 2. Let $F^R(a,z)$ be the remainder of $F(Q,R)$ after introducing several stages of the cascade represented by $\varphi(a,z)$. Let the set of variables a be divided into two parts, those to be removed from the support, a_r, and those to remain in the support, a_s. The implementation with variables a_r removed from the support exists iff

$$\chi(z) = \exists a_s\, [\exists a_r\, Q^R(a,z)\ \&\ \exists a_r\, R^R(a,z)] \neq 1.$$

Taking any assignment z_0 of variables z such at $z_0 \in \overline{\chi(z)}$ gives one feasible implementation.

Using this theorem we can find an implementation of several stages of the wave cascade at a time. When z_0 is computed, we set $z = z_0$ in $F^R(a,z) = (Q^R(a,z), R^R(a,z))$ and derive the incompletely specified implementation of the remainder $F^{RI}(a) = (Q^R(a,z_0), R^R(a,z_0))$, which replaces $F(a)$ in the next iteration of the algorithm.

The proposed algorithm does not work for complex functions for the following reasons. The representation of several stages of the cascade (in the case when one stage does not lead to the support reduction) may become too large. The runtime need to compute $\chi(z)$ in Theorem 2 requires three BDD traversals and tends to timeout for complex functions.

6 Experimental results

Implementation of the algorithm in Section 5 is not completed. In this section, we provide an upper bound on the number of generalized Maitra terms in the reversible wave cascade. It was shown in Section 2 that ESOPs of Boolean functions can be mapped into reversible wave cascades by implementing each product in the ESOP as a generalized Maitra term.

The upper bound on the number of stages in the reversible wave cascade is computed for selected MCNC benchmarks as the number of terms in the heuristically minimized ESOP of the Boolean functions [14]. Note that this upper bound works for multi-output functions.

Table 1. Upper bound on the number of stages in reversible wave cascades.

Benchmark			Upper Bound
Name	Inputs	Outputs	
5xp1	7	10	31
9sym	9	1	51
add6	12	7	127
addm4	9	8	89
b12	15	9	28
clip	9	5	63
ex7	16	5	81
f51m	8	8	31
in7	26	10	35
intb	15	7	268
life	9	1	48
m181	15	9	29
m4	8	16	76
max512	9	6	82
rd53	5	3	14
rd73	7	3	36
rd84	8	4	58
ryy6	16	1	40
sao2	10	4	28
seq	41	35	246
sym10	10	1	79
t3	12	8	24
t481	16	1	13
vg2	25	8	184
z4	7	4	29
Average	13.0	7.0	71.6

The first three columns in Table 1 characterize the benchmarks. “Names” gives the benchmarks name. “Inputs” gives the number of inputs. “Outputs” gives the number of outputs. “Upper Bound” gives the maximum number of stages in the reversible wave cascade.

7 Conclusions

This paper presents an algorithm to generate reversible wave cascades implementing incompletely specified Boolean functions. Minimizing garbage bits is the main challenge of reversible logic synthesis. The remarkable property of the presented method, compared to other reversible synthesis methods, is that it creates at most one constant input and no additional garbage outputs.

References

[1] W.C. Athas and L."J." Svensson. *Reversible Logic Issues in Adiabatic CMOS.* Exploratory Design Group, University of Southern California, Information Sciences Institute, Marina del Rey, CA.

[2] C. Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development*, 17, 1973, pp. 525-532.

[3] R. E. Bryant. Graph-Based Algorithms for Boolean function Manipulation. *IEEE Trans. Computers*, C-35(8), pp. 677-691, 1986.

[4] R. Cuykendall and D. McMillin. Control-Specific Optical Fredkin Circuits. *Applied Optics*, 26, pp. 1959-1963, 1987.

[5] A. De Vos. Design of Reversible Logic Circuits by Means of Control Gates. *Proc. Patmos 2000 Conference*, Goettinge (*Springer Lecture Notes in Computer Science*, Vol. 1918), pp. 255-264.

[6] A. De Vos. Control Gates as Building Blocks for Reversible Computers. *Proc. Patmos 2001 Conference*, Yverdon.

[7] E. Fredkin and T. Toffoli. Conservative Logic. *Int. Journal of Theor. Phys.*, 21 (1982), pp. 219-253.

[8] P. Kerntopf. A Comparison of Logical Efficiency of Reversible and Conventional Gates. *Proc. IWLS'01.*

[9] A. Khlopotin, P. Kerntopf, M. Perkowski. Reversible Logic Synthesis by Iterative Compositions. *Proc. IWSL'02.*

[10] G. Lee. Logic synthesis for cellular architecture FPGA using BDD. *Proc. ASP-DAC*, pp. 253-258, 1997.

[11] G. Lee and R. Drechsler. ETDD-Based Synthesis of Term-Based FPGAs for Incompletely Specified Boolean Functions. *Proc. ASP-DAC*, pp.75-80, 1998.

[12] G. Lee, R. Drechsler, and M. Perkowski. ETDD-Based Synthesis of Two-Dimensional Cellular Arrays for Multi-Output Incompletely Specified Boolean Functions," *IEE Proc. Comput. Digit. Tech*, Vol. 146, No. 6, November 1999, pp. 302 - 308.

[13] K. K. Maitra. Cascaded switching networks of two-input flexible cells. *IRE Trans. Electron. Comput.*, pp, 136-143. 1962.

[14] A. Mishchenko and M. Perkowski. Fast Heuristic Minimization of Exclusive Sum-of-Products. *Proc. Reed-Muller Workshop, 2001*, Starkville, Mississippi, pp. 242-250.

[15] M. Perkowski, E. Pierzchala. New Canonical Forms for Four-Valued Logic. *Technical Report, Electrical Engineering Department, PSU, 1993.*

[16] M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al-Rabadi, A. Coppola, A. Buller, X. Song, Md. M. Khan, S. Yanushkevich, V. Shmerko, and M. Chrzanowska-Jeske. A General Decomposition for Reversible Logic. *Proc. RM 2001*, Missisipi State University, August 2001.

[17] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, Alan Coppola, B. Massey. Regularity and Symmetry as a Base for Efficient Realization of Reversible Logic Circuits. *Proc. IWLS 2001.*

[18] M. A. Perkowski, A. Sarabi, F. R. Beyl. Universal XOR Canonical Forms of Switching Functions. *Proc. of IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design.* Hamburg, Germany, September 16-17, pp. 27 - 32, 1993.

[19] P. Picton. A Universal Architecture for Multiple-valued Reversible Logic. *MVL Journal,* 5, 2000, pp.27-37.

[20] J. Preskill. Lecture Notes in Quantum Computing. http://www. Theory.caltech.edu/~preskill/ph229

[21] A. Sarabi, N. Song, M. Chrzanowska-Jeske, M. A. Perkowski. A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays. *Proc. DAC'94,* pp. 321 - 326.

[22] N. Song, M. Perkowski. Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Functions. *IEEE Trans. CAD,* Vol. 15, No. 4, April 1996, pp. 385-395.

[23] T. Toffoli. Reversible Computing. In *Automata, Languages and Programming*, Springer Verlag, 1980, pp. 632- 644.