Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

Spring 5-29-2015

Reversible Circuits Synthesis Based on EXOR-sum of Products of EXOR-sums

Linh Hoang Tran

Portland State University

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

Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Tran, Linh Hoang, "Reversible Circuits Synthesis Based on EXOR-sum of Products of EXOR-sums" (2015).
Dissertations and Theses. Paper 2302.

<https://doi.org/10.15760/etd.2299>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.

Reversible Circuits Synthesis

Based on EXOR-sum of Products of EXOR-sums

by

Linh Hoang Tran

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Electrical and Computer Engineering

Dissertation Committee:
Marek Perkowski, Chair
Xiaoyu Song
Fu Li
John Caughman

Portland State University
2015

Abstract

Power dissipation in modern technologies is an important matter and overheating is a severe concern for both manufacturer (impossibility of introducing new and smaller scale technologies and limited temperature range for operating the product) and customer (power supply, which is especially important for mobile systems). One of the main profits that reversible circuit carries is theoretically the zero power dissipation in the sense that it is independent of underlying technology; irreversibility means heat generation. In the other words, reversible circuits may offer a feasible solution in the future that will aid certain reduction of the power loss.

Reversible circuits are circuits that do not lose information during computation. These circuits can create unique output vector from each input vector, and vice versa, that is, there is a one-to-one mapping between the input and the output vectors. Historically, the reversible circuits have been inspired by theoretical research in low power electronics as well as practical progress of bit-manipulation transforms in cryptography and computer graphics. Interest in reversible circuit is also sparked by its applications in several up-to-date technologies, such as Nanotechnology, Quantum Computing, Optical Computing, Quantum Dot Cellular Automata, and Low Power Adiabatic CMOS. However, the most important application of reversible circuits is in Quantum Computing.

Logic synthesis methodologies for reversible circuits are very different from those for classical CMOS and other technologies. The dissertation introduces a new concept of reversible logic circuits synthesis based on EXOR-sum of Products-of-EXOR-sums

(EPOE). The motivation for this work is to reduce the number of the multiple-controlled Toffoli gates as well as the numbers of their inputs. To achieve these reductions the research generalizes from the existing 2-level AND-EXOR structures (ESOP) commonly used in reversible logic to a mixture of 3-level EXOR-AND-EXOR structures and ESOPs. The approaches can be applied to reversible and permutative quantum circuits to synthesize both completely and incompletely specified single-output functions as well as multiple-output functions.

This dissertation describes the research intended to examine the methods to synthesize reversible circuits based on this new concept. The examinations indicate that the synthesis of reversible logic circuits based on EPOE approach produces circuits with significantly lower quantum costs than the common ESOP approach.

Table of Contents

Abstract	i
List of Tables	v
List of Figures	vi
Chapter 1: INTRODUCTION.....	1
1.1 Introduction:	1
1.2 Goals:	4
Chapter 2: BACKGROUND AND LITERATURE REVIEW ON BASIC REVERSIBLE GATES.....	5
2.1 Affine Linear Function:.....	5
2.2 Reversible Logic Circuit:	6
2.3 Quantum Cost Metric:.....	9
2.4 POE-terms of the same support family:	11
2.5 Fractional Covering Criterion Analysis:	12
2.5.1 $\frac{3}{4}$ covering criterion:	12
2.5.2 $\frac{2}{3}$ covering criterion:	13
Discussion:.....	15
Chapter 3: SYNTHESIS OF EPOE CIRCUITS FOR COMPLETELY SPECIFIED SINGLE OUTPUT BOOLEAN FUNCTIONS	16
3.1 Template matching method based on the library of templates:	16
3.1.1 Generation of Template Libraries for N-variable Functions:	17
3.1.2 EPOEM-1s using single expression template library:	19
3.1.3 EPOEM-1f using full expression template library:	24
3.1.4 Experimental Results:.....	26
3.2 Algebraic method based on factorization:.....	30
3.2.1 Proposition 1:.....	30
3.2.2 Proposition 2:.....	33
3.2.3 The EPOEM-2 Algorithm:.....	35

3.2.4	Experimental results:	43
3.3	Discussion about two methods:.....	46
Chapter 4: SYNTHESIS OF EPOE CIRCUITS FOR INCOMPLETELY SPECIFIED SINGLE OUTPUT BOOLEAN FUNCTIONS		48
4.1	EPOEM-1-DC Algorithm:	50
4.2	Examples of EPOEM-1-DC:.....	51
4.3	EPOE-1-DC-tree algorithm:.....	54
4.4	Examples of EPOEM-1-DC-tree application:.....	55
4.5	Experimental Results:	62
Chapter 5: SYNTHESIS OF EPOE CIRCUITS FOR MULTIPLE OUTPUT BOOLEAN FUNCTIONS		65
5.1	EPOEM-MO-1:	65
5.1.1	Phase 1:	66
5.1.2	Phase 2:	69
5.1.3	Phase 3:	73
5.1.4	Experimental Results:	78
5.2	EPOEM-MO-2:	80
5.2.1	Synthesis based on Exclusive-OR Lattices:.....	80
5.2.2	EPOEM-MO-2 Algorithm:	83
5.2.3	Experimental Results:	90
5.3	Discussion about EPOEM-MO-1 and EPOEM-MO-2 methods:.....	93
Chapter 6: APPLICATIONS OF SYNTHESIS OF EPOE CIRCUITS, QUANTUM AUTOMATA AND REVERSIBLE HARDWARE CLASSIFIERS		94
Chapter 7: CONCLUSION		102
WORKS CITED		104
Appendix A: Glossary.....		115

List of Tables

Table 1. Quantum costs of fundamental reversible gates according to Maslov.	10
Table 2. Examples of POE template expressions for functions of three variables.	19
Table 3. Four and five input variable functions synthesized with EXORCISM-4 vs. EPOEM vs. REVLIB	27
Table 4. Six and more input variable functions synthesized with EXORCISM-4 vs. EPOEM-1S	29
Table 5. Benchmark functions synthesized with EXORCISM-4 (3) vs. EPOEM-1S (1) vs. EPOEM-2 (2) vs. EXACT MINIMUM ESOP (4) vs. EXACT MINIMUM EPOE EPOE-EXACT (5)	45
Table 6. Comparison of EPOEM-1 and EPOEM-2 method	47
Table 7. Incompletely specified single output benchmark functions	62
Table 8. Benchmark functions synthesized with EPOEM-1-DC vs. EPOEM-1-DC-tree	63
Table 9. Benchmark functions synthesized with EPOEM-1-DC vs. EPOEM-1S vs. EXORCISM-4.....	64
Table 10. Experimental results for quantum cost comparison.....	78
Table 11. EPOE expression and quantum cost of each function in EXOR-lattice and the function ($A \oplus D$)	86
Table 12. Sorted functions list base on the quantum costs of the functions	86
Table 13. Quantum cost comparison between the three methods of creating EXOR lattice	89
Table 14. Experimental results for quantum cost comparison.....	92
Table 15. Example quantum automation for synthesis with EPOEM-MO-2	96
Table 16. States encoding with 1-out-of-4 code	96
Table 17. Example system from Table 14 with encoded states	96
Table 18. Example system for synthesis with encoded states	97
Table 19. List of EPOE expressions and their quantum costs of each node.....	98
Table 20. Sorted functions list by cost.....	99

List of Figures

Figure 1. Karnaugh map that illustrate the POE expression $(a \oplus b)(c \oplus d)$ which realizes a four minterm function	4
Figure 2. A reversible circuit with a NOT, CNOT, and Toffoli gate	6
Figure 3. EPOE synthesis of function $g = (a \oplus b \oplus 1)(d \oplus 1) \oplus (b \oplus 1)(c \oplus 1)(a \oplus d \oplus 1)$	8
Figure 4. Groupings: a. without 0-terms for $\frac{3}{4}$ -majority 3-cube	13
Figure 5. Karnaugh map that illustrate alternative ESOP syntheses of the function F_1 : a) for expression $cef \oplus ac'def$ and b) for expression $ef \oplus c'ef \oplus ac'def$	14
Figure 6. Karnaugh map that illustrate alternative ESOP syntheses of function F_2 : a) for expression $ef \oplus c'd'ef \oplus a'b'c'def$ and b) for expression $ef \oplus c'ef \oplus c'def \oplus a'b'c'def$	14
Figure 7. Karnaugh maps that illustrate template selection for EPOEM-1s and EPOEM-1f.	20
Figure 8. Karnaugh maps that illustrate the remainder functions generated from function F_4 for two templates in EPOEM-1s	22
Figure 9. Karnaugh maps that illustrate the remainder function evolution in EPOEM-1s	22
Figure 10. Two realizations of the EPOE circuits for expression $F_4(a,b,c,d) = (a \oplus d) \oplus (a \oplus b \oplus 1)(d \oplus 1) \oplus (b \oplus 1)(c \oplus 1)(a \oplus d \oplus 1)$. a) Circuit with no garbage line but with mirror gates b) Circuit with no mirror gates but with garbage lines.....	23
Figure 11. Standard ESOP-like circuit realization of function $F_4(a, b, c, d) = b'cd' \oplus a'd \oplus ab'c$ with only Toffoli and NOT gates. The quantum cost is 39.	24
Figure 12. Circuit for function F_4 produced by EPOEM-1f algorithm.....	26
Figure 13. Frequency distribution of quantum cost results from 50 random runs using EPOEM-1s to synthesize functions: a) 4sf, b) 5ex3, c) 5majority, d) 6sym	29
Figure 14. Karnaugh maps that illustrate applications of Lemma 2	34
Figure 15. Circuit realization of function F_7 using EPOEM-2 algorithm.....	42
Figure 16. Standard ESOP-like circuit realization of function $F_7 = a' \oplus b'e \oplus b'c' \oplus cde \oplus a'bc'e' \oplus ac'd'e' \oplus a'b'cde$ with only Toffoli and NOT gates.....	42
Figure 17. Frequency distribution of quantum cost results from 50 random runs using EPOEM-2 to synthesize functions: a) 5ex3, b) 6sym, c) 9sym	46
Figure 18. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC.	52
Figure 19. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC.	52

Figure 20. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC.....	53
Figure 21. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC.....	53
Figure 22. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC-tree at $level = 1$	58
Figure 23. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC-tree at $level = 2$ and $level = 3$	58
Figure 24. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC-tree at $level = 2$	60
Figure 25. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC-tree.....	60
Figure 26. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC-tree at $level = 3$	61
Figure 27. Frequency distribution of quantum cost results from 50 random runs using EPOEM-1-DC to synthesize functions: a) dc42, b) dc51, c) dc52 and d) dc61	64
Figure 28. Truth table illustrates the transformation from Example 1.	68
Figure 29. Truth table illustrates the transformation from Example 2.	69
Figure 30. Circuit realization of Example 1 produced by EPOEM-MO-1 algorithm without shared-POE-term phase	74
Figure 31. Circuit realization of Example 1 produced by EPOEM-MO-1 algorithm with shared-POE-term phase	75
Figure 32. Initial POE term list.....	76
Figure 33. POE-list and its sub-lists	77
Figure 34. Circuit realization of Example 2 produced by EPOEM-MO-1 algorithm with shared-POE-term phase	77
Figure 35. Frequency distribution of quantum cost results from 50 random runs using EPOEM-MO-1 to synthesize functions: a) Perk01, b) rd53, c) cm82a	79
Figure 36. Flattened EXOR Tree (Special case of EXOR Lattice)	80
Figure 37. Karnaugh map that illustrate output function A, B and C	82
Figure 38. Karnaugh map that illustrate logical functions in their EXOR Lattice form and Coverings are first found for A', F1, and F3 by using EXORCISM-4.....	82
Figure 39. Circuit realization of: a) Function A, b) Function F1, c) Function F2 and d) Functions: A, F1 and F3 in cascade.	82
Figure 40. Circuit realization of: a) Function F2 from function F1 and function F3, b) Function B from function A and function F1, c) Function C from function B and function F2	83
Figure 41. The complete circuit realization of A, B and C.....	83
Figure 42. Example function that used for EPOEM-MO-1 synthesis	84

Figure 43. EXOR-lattice of function given in Figure 42.	85
Figure 44. Circuit realization of function given in Figure 42 in the case of not using the function $(A \oplus D)$	87
Figure 45. Circuit realization of function given in Figure 42 in the case of using function $(A \oplus D)$	88
Figure 46. Frequency distribution of quantum cost results from 50 random runs using EPOEM-MO-2 to synthesize functions: a) Perk01, b) rd53, c) cm82a and d) rd73	91
Figure 47. Karnaugh maps that illustrate the the case of EXORing between “1-minterm” with a “don’t care term” when EXORing function A and function B	97
Figure 48. Circuit realization of permutative quantum automaton from Table 14. A feedback loop from outputs A, B, C, D to inputs a, b, c, d, respectively is not shown. This loop includes potentially some flip-flops. Signals e and f are primary inputs and signal E is an output of the machine	101

Chapter 1: INTRODUCTION

1.1 Introduction:

Management of energy loss is a significant concern in digital logic design. An increasingly large fraction of this energy loss happens because of the non-ideality of physical switches and devices. Progresses in VLSI technology and the use of new fabrication processes over the last few decades have rendered the heat loss and dissipation problem more complex in integrated circuits (IC). The essential energy loss, resulting from the irreversibility of information processing as confirmed by Landauer's principle [27], may be reduced by using a reversible computing model, which is information lossless, i.e., when the input information fed to a computing system is uniquely determinable from the observed output information [28, 29, 30]. With the continuing exponential growth of integrated circuits (IC) technology as predicted by the Moore's Law, energy loss in non-reversible designs is likely to become more dominant, and reversible logic may offer a feasible solution in the future with the availability of newer technologies for its implementation, for e.g., spintronics [31]. Reversibility can also be implemented to some extent by using classical CMOS and adiabatic charging techniques that support reduction of energy loss and following improvement of power efficiency, without deploying voltage scaling [32]. The reversible circuits have been shown to have many applications to the evolving field of quantum computation [33, 34, 35], which has potential of solving some exponentially hard problems in polynomial time.

The objective of the research in quantum and reversible logic synthesis is to create efficient algorithms that convert high-level specifications of reversible and quantum permutative functions¹ to sequences of basic reversible or quantum gates that minimize associated cost. This synthesis problem is difficult, so researchers have applied different approaches based on classical logic synthesis and various mathematical concepts. Therefore there exist several types of algorithms for synthesis of reversible circuits and quantum permutative circuits, including: (1) cycle-based methods [12, 45, 46], (2) group-theory based methods [15, 22, 23, 24], (3) transformation-based methods like MMD [2, 7, 25], (4) BDD-based methods [16, 26], and (5) Exclusive-Or-Sum-of-Products (ESOP) based methods [3, 5, 6, 10, 11, 82, 83, 84, 85]. The reversible ESOP-based methods have two variants, those that start from arbitrary reversible specifications and those that start from arbitrary (reversible or not) specifications but realize a circuit in which one ancilla line is added for every output. The latter ESOP variant has utility in mapping irreversible functions to reversible functions and has been studied in [10, 11] using a quantum cost metric. This research follows the latter ESOP variant and introduces a new concept of gate structure called EXOR-sum of Products-of-EXOR-Sums (EPOE) to which specifications are mapped. The dissertation presents algorithms that convert Boolean input specifications to EPOE structures. Next, circuits synthesized with the EPOE minimizer EPOEM are compared with the circuits from the ESOP minimizer EXORCISM-4 [9] and ESOP-based methods from [3, 82, 83, 84, 85] using a quantum cost metric. Three variants of the EPOEM

¹ A quantum permutative function is a binary reversible function where binary gates such as Toffoli and Feynman are internally realized with quantum primitives such as Controlled-V. The costs for reversible circuits and quantum permutative circuits differ considerably. This difference affects the choice of the structures to which functions are mapped and their respective synthesis algorithms.

algorithm, EPOEM-1s, EPOEM-1f and EPOEM-2, used for synthesis of reversible circuits of a complete specified single output function, are introduced. As well as EPOEM-1-DC, EPOEM-MO-1 and EPOEM-MO-2, which are used for synthesis of reversible circuits of an incompletely specified single output function and multiple output function respectively are also introduced herein. The three variants, EPOEM-1s, EPOEM-1f and EPOEM-2, are used to investigate the trade-offs between reversible circuit dimension and latency.

EPOE's main advantage over ESOP synthesis is that it solves the even/odd covering problem with compound Products-of-EXOR-Sums (POE), or pseudoproduct [77, 79, 80, 81], gates rather than the Products-of-literals gates. This produces lower quantum cost circuits than EXORCISM-4 for a majority of tested benchmark functions. The EPOEM programs achieve lower quantum costs by simplifying expressions of multiple high Hamming Distance minterms into new expressions that EXORCISM-4 is incapable of producing. In principle EPOE synthesis selects the best fractional covering EXOR-sums of literals and ANDs them together to obtain POE terms. For instance, Figure 1 shows how a POE synthesis of a function of four minterms compactly expresses an AND of two relatively simple EXOR-sums of literals. By comparing EPOE solutions with ESOP solutions for the same functions we observed that EPOE reduced the number of multiple control Toffoli gates and their number of inputs. This means that quantum costs of circuits that result from EPOE algorithms are generally lower and never worse than those that come from ESOPs.

Figure 1. Karnaugh map that illustrate the POE expression $(a \oplus b)(c \oplus d)$ which realizes a four minterm function

1.2 Goals:

Described in this dissertation is a research project that investigates methods that synthesize the reversible circuit based on EPOE. My objectives are threefold. First, I examine the methods that convert a completely specified single output Boolean function of N input variables $(a, b, c, d, \dots)$ into EPOE expression. Second, I extend these methods to synthesize incompletely specified single-output functions (i.e. functions with don't cares). Third, I investigate how to synthesize the multiple output function for quantum and reversible circuits with EPOE type circuits.

Chapter 2: BACKGROUND AND LITERATURE REVIEW ON BASIC REVERSIBLE GATES

2.1 Affine Linear Function:

It is well known that Reed-Muller expansions are used in logic synthesis and design of highly testable circuits [68, 69, 70, 71, 72]. In the area of reversible computing, Younnes and Miller in their work [73] have introduced the techniques for representation of quantum Boolean circuits using Reed-Muller expansions. Similarly, as stated by Saeedi [18], to specify and synthesize circuits, algebraic formulas based on Positive polarity Reed-Muller (PPRM) expansion can be applied. PPRM expansion uses only un-complemented variables and can be derived from the EXOR-Sum-of-Products (ESOP) description (or DSOP – Disjunctive Sum of Products specification) by replacing a' with $a \oplus 1$ for a complemented variable a and then performing standard simplifying transformations of Boolean algebra. The PPRM expansion of a function is canonical and is defined as follows.

$$f(x_1, x_2, \dots, x_n) = c_0 \oplus c_1 x_1 \oplus \dots \oplus c_n x_n \oplus c_{12} x_1 x_2 \oplus \dots \oplus c_{n,n-1} x_{n-1} x_n \oplus \dots \oplus c_{12\dots n} x_1 x_2 \dots x_n$$

By Harrison [74] and De Vos [15], a function $f(x_1, x_2, \dots, x_n)$ is affine linear if and only if its Reed-Muller expansion contains only terms with either zero or one coefficients, it means an affine functions is an EXOR of variables and possible a constant 1:

$$f(x_1, x_2, \dots, x_n) = c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus \dots \oplus c_n x_n$$

Each of the coefficients c_i can take one of the two values: 0 or 1. So, there are 2^{n+1} different affine linear functions of n arguments.

For examples: With $n = 3$, a function $f(x_1, x_2, x_3)$ has 16 different affine linear functions. These are:

0	1
x_1	$x_1 \oplus 1$
x_2	$x_2 \oplus 1$
x_3	$x_3 \oplus 1$
$x_1 \oplus x_2$	$x_1 \oplus x_2 \oplus 1$
$x_1 \oplus x_3$	$x_1 \oplus x_3 \oplus 1$
$x_2 \oplus x_3$	$x_2 \oplus x_3 \oplus 1$
$x_1 \oplus x_2 \oplus x_3$	$x_1 \oplus x_2 \oplus x_3 \oplus 1$

Each of the terms in a Product of EXOR-sum is an affine linear function.

2.2 Reversible Logic Circuit:

An arbitrary reversible Boolean function is a one-to-one and onto function. Assuming N input and N output wires, it is the mapping from 2^N to 2^N binary combinations. In the physical implementation of classical reversible logic, each input/output pair is typically called a line or a wire, whereas in quantum logic it is called a qubit. A reversible circuit schematic representation with three lines is shown in Figure 2. In the figure input signal combinations propagate from left to right through horizontal lines and can be modified as they pass through a cascade of reversible gates.

Figure 2. A reversible circuit with a NOT, CNOT, and Toffoli gate

The fundamental or classical reversible gates correspond to the following Boolean functions:

$$\text{NOT: } (x_1) \rightarrow (x_1 \oplus 1) \quad (1)$$

$$\text{CNOT: } (x_1, x_2) \rightarrow (x_1, x_2 \oplus x_1) \quad (2)$$

$$\text{Tofolli: } (x_1, x_2, x_3) \rightarrow (x_1, x_2, x_3 \oplus x_1x_2) \quad (3)$$

Multiple-control $N \times N$ Tofolli:

$$(x_1, x_2, \dots, x_{N-1}, x_N) \rightarrow (x_1, x_2, \dots, x_{N-1}, x_N \oplus x_1x_2 \dots x_{N-1}) \quad (4)$$

For simplicity the ESOP-based reversible circuit synthesis methods considered here will be restricted to a subset of arbitrary reversible Boolean functions in which $N - 1$ lines $x_1, x_2, \dots, x_{N-1}$ are treated as inputs which remain unchanged at the circuit's output and one ancilla line x_N , initialized to 0, is used as an output. This single output function can be expressed in the following form:

$$(x_1, x_2, \dots, x_{N-1}, x_N = 0) \rightarrow (x_1, x_2, \dots, x_{N-1}, x_N \oplus f(x_1, x_2, x_3, \dots, x_{N-1})) \quad (5)$$

As mentioned above the ancilla output line to produce function f is initialized to $x_N = 0$.

The family of arbitrary reversible Boolean functions realized using only CNOT and NOT gates is known as affine-linear reversible circuits [15]. This family can be represented compactly in the form $Y = MX \oplus B$ where X is a vector of N Boolean inputs representing reversible circuit lines, M is an N by N Boolean coefficient matrix which is invertible under $GF(2)$, and B is a vector of N Boolean constants. In this treatment the product of matrix M with vector X uses AND for multiplication and EXOR for addition.

Consequently, elements of the vector Y are linear functions of X with a constant term from vector B of the following form as the EXOR-sum of literals:

$$y_k = a_{k1}x_1 \oplus a_{k2}x_2 \dots \oplus a_{kn}x_n \oplus b_k \quad (6)$$

If the above equation is reduced by one input to an $N-1 \times N-1$ affine-linear reversible circuit, then the POE function f in (5) is expressed as a product of EXOR-sum of literals:

$$(x_1, x_2, \dots, x_{N-1}, x_N) \rightarrow (x_1, x_2, \dots, x_{N-1}, x_N \oplus y_k y_l \dots y_m) \quad (7)$$

EPOE-synthesized circuits consist of one or more POE expressions, each realized using three components: an $N-1 \times N-1$ affine-linear reversible circuit which modifies only input lines, a single fundamental reversible gate targeting the output line which can have a size ranging from a multiple-control $N \times N$ Toffoli to the NOT gate, and an inverse circuit, or mirror, of the $N-1 \times N-1$ affine-linear reversible circuit. An example EPOE circuit follows.

Figure 3. EPOE synthesis of function $g = (a \oplus b \oplus 1)(d \oplus 1) \oplus (b \oplus 1)(c \oplus 1)(a \oplus d \oplus 1)$.

Figure 3 shows an EPOE circuit with two POE gates. In this circuit each POE gate includes an EXOR-sum of literals at the left and its mirror EXOR-sum of literals at the right which returns the control lines to their original values a , b , c , and d . By eliminating subsequent pairs of identical gates one can optimize the circuit in Figure 3. This transformation is permissible because the NOT and CNOT gates that are used in the mirror circuits are self-inverses.

2.3 Quantum Cost Metric:

This dissertation uses the well-known Maslov's quantum cost [4, 75, 76]. In this approach, the quantum cost (QC) of a circuit is defined by the total sum of costs of gates in the circuit where the cost of each gate (NOT, CNOT, Toffoli, $N \times N$ Toffoli) is defined in Table 1. This particular metric is used in the majority of papers written by authors from the reversible and quantum computing community as found in Revlib's page [44] and Maslov's benchmark page [45].

The following important remark is here in order. Different areas of research apply various criteria how to evaluate the quality of numerical results of their optimization algorithms. This applies particularly to logic synthesis, the area of research that started in 1950's and flourished since 1980. Some logic synthesis authors with background in theoretical computer science use mathematical calculations of lower and upper bounds for their algorithms but it is known from practice that in most cases these bounds tell very little about the quality of algorithms when run on test examples taken from real functions. These ideas are then not treated seriously by reviewers of top journal and conferences that publish results in logic synthesis. In contrast, several past authors were using randomly generated functions for comparison of their tool with other tools. But it is known that randomly generated functions are statistically much more difficult than functions taken from real engineering problems. This is why the idea of standard benchmark Boolean functions was created by IBM and University of California at Berkeley and several of such sets of benchmark functions were developed by ISCAS, MCNC, U.C. Berkeley, Maslov [45] and Revlib [44]. These are some important or typical Boolean functions, or components such

as adders or Finite State Machines taken from industrial companies. Comparison on standard benchmarks is a standard and popular method used in last 15 years in this area. This method will be also followed by me in this dissertation. In addition, I will do some statistical analysis on randomized algorithms and I will compare my results with the exact ESOP minimum solution. Concluding, using Maslov Quantum Costs, I will apply several evaluation methodologies to confirm the high quality of my solutions.

Table 1. Quantum costs of fundamental reversible gates according to Maslov.

Gate	Cost
NOT	1
CNOT	1
Toffoli	5
N×N Toffoli	$2^N - 3$

As we can see from Table 1, the quantum cost of Toffoli gate increases exponentially as the number of inputs increase. So the motivation of all the methods presented in this dissertation is to reduce the number of multiple controlled Toffoli gates and the total number of inputs and thus the quantum cost. Concluding, in classical circuit synthesis EXOR gate is expensive and the AND gate is inexpensive, so the synthesis methods are geared to these costs, which means that EXOR gates are used only when necessary. In quantum circuit synthesis the EXOR gate (Feynman gate) is very inexpensive and the Toffoli gate (AND) is expensive. Especially expensive is a multi-input Toffoli. Therefore when synthesizing quantum circuits the synthesis algorithm must be primarily based on using EXOR as a combining operator (as in ESOP) but also in reducing the sizes of Toffoli gates by smart factorization that involves many EXOR gates (as in EPOE).

One may also ask, how this dissertation is related to quantum circuits as there is very little “quantum ideas” here. We are not involved in this dissertation with classical

quantum computing concepts such as superposition or entanglement, dense coding or teleportation. The link of the research presented here to quantum computing is **only** through quantum costs. We are not minimizing the reversible functions with respect to CMOS cost but to quantum cost. This single assumption completely changes the synthesis approaches. The circuits discussed in this dissertation are not general quantum circuits that operate in Hilbert Space, our circuits are the permutative quantum circuits which means, circuits specified by permutative binary matrices (subsets of Hilbert space) but we use gates that are implementable in quantum and thus allow to have superpositions and entanglements in them (we are not discussing internal structures of Toffoli and other gates here. For description of realization of quantum gates see [34]). One has to remember that from the practical point of view, the absolute most of gates used in quantum algorithms are permutative gates and only few other, non-permutative quantum gates, such as for instance Hadamard. For instance in the famous Grover Algorithm, the problem that is solved is reduced to the design of a quantum circuit called an oracle. The oracle is exercised with superposed states coming from many Hadamard gates. This oracle is however a purely permutative circuit and can be built using methods presented in this dissertation. Other gates used in Grover algorithm, such as Hadamard or Controlled-Z are standard and their synthesis belongs to the area of physics [34] and not to the area Computer Aided Design of Quantum Circuits, the topic of this dissertation.

2.4 POE-terms of the same support family:

The polarity of the EXOR-sum of literals is defined as the constant portion of the expression. For instance, $(x_1 \oplus x_2)$ has a polarity of 0 and $(x_1 \oplus x_2 \oplus 1)$ has a polarity of 1.

The expressions $(x_1 \oplus x_2)$ and $(x_1 \oplus x_2 \oplus 1)$ are two polarities of the same EXOR-sum $(x_1 \oplus x_2)$

The support family of a POE expression p is defined as the set of POE expressions with the same set of EXOR-sums as in p but with all of their possible polarities. For instance, the support family of POE $(x_1 \oplus x_2)(x_3 \oplus x_5)$ is the following:

$$\left\{ \begin{array}{l} (x_1 \oplus x_2)(x_3 \oplus x_5) \\ (x_1 \oplus x_2 \oplus 1)(x_3 \oplus x_5) \\ (x_1 \oplus x_2)(x_3 \oplus x_5 \oplus 1) \\ (x_1 \oplus x_2 \oplus 1)(x_3 \oplus x_5 \oplus 1) \end{array} \right.$$

Thus the first and second POE expressions above differ only in the polarity of EXOR-sum $(x_1 \oplus x_2)$

2.5 Fractional Covering Criterion Analysis:

This subsection reviews the fractional covering criteria for modulo-2 sum expressions, for e.g. the $\frac{3}{4}$ covering criterion from [86] and $\frac{2}{3}$ covering criterion from [88].

2.5.1 $\frac{3}{4}$ covering criterion:

As introduced by Anh Tran in [86] with the examples' results for a $\frac{3}{4}$ -majority 3-cube and a $\frac{3}{4}$ -majority 4-cubes are shown in Figure 4. There are two different ways of grouping, with and without 0-terms. By comparing the two different ways of grouping, it is seen that, if 0-terms are incorporated with a $\frac{3}{4}$ -majority m-cube to form an m-cube, the total number of groupings will be equal to or less than that resulted from grouping only 1-terms in the $\frac{3}{4}$ -majority m-cube. More comparisons of patterns in maps used to determine an empirical selection criterion for product groups can be found in [87]. Based on the above discussions and studies, $\frac{3}{4}$ -majority cubes are instituted as a selection criterion.

Figure 4. Groupings: a. without 0-terms for $\frac{3}{4}$ -majority 3-cube
b. with 0-terms for $\frac{3}{4}$ -majority 3-cube in a
c. without 0-terms for $\frac{3}{4}$ -majority 4-cube
d. with 0-terms for $\frac{3}{4}$ -majority 4-cube in c

2.5.2 $\frac{2}{3}$ covering criterion:

This subsection presents another analysis leading to the formulation of the $\frac{2}{3}$ covering criterion for modulo-2 sum expressions as introduced by our PSU team in [88]. In order to do this, two similar functions, F_1 and F_2 shown in Figure 5 and Figure 6 respectively, will each be synthesized twice and judged strictly on AND gate counts. The function F_1 is synthesized in Figure 5a using a greater than $\frac{2}{3}$ fractional criterion. Under this restriction the most efficient covering selection is comprised of disjoint groups cef and $ac'def$, realized by one three-input AND gate and one five-input AND gate. This results in the disjoint ESOP $F_1 = cef \oplus ac'def$. In contrast in Figure 5b the $\frac{2}{3}$ covering criterion is violated. Consequently the synthesis in Figure 5b has more steps and the corresponding circuit is realized with one two-input AND gate, one three-input AND gate, and one five-input AND gate. This results in the non-disjoint ESOP $F_1 = ef \oplus c'ef \oplus ac'def$. Violating the $\frac{2}{3}$ covering criterion in the choice of product ef leads to an additional AND gate.

Figure 5. Karnaugh map that illustrate alternative ESOP syntheses of the function F_1 : a) for expression $cef \oplus ac'def$ and b) for expression $ef \oplus c'ef \oplus ac'def$.

The function F_2 is synthesized in Figure 6a using a “greater than 2/3” covering criterion. Under this restriction the most efficient covering selection is comprised of non-disjoint groups ef , $c'd'ef$, and $a'b'c'def$ which requires one two-input AND gate, one four-input AND gate, and one six-input AND gate. This results in the non-disjoint ESOP $F_2 = ef \oplus c'd'ef \oplus a'b'c'def$. In contrast in Figure 6b the 2/3 covering criterion is violated. Consequently the synthesis in Figure 6b has more steps and leads to a circuit with one two-input AND gate, one three-input AND gate, one four-input AND gate, and one six-input AND gate. This results in the non-disjoint ESOP $F_2 = ef \oplus c'ef \oplus c'def \oplus a'b'c'def$. Again, violating the 2/3 covering criterion in the choice of product $c'ef$ leads to an additional AND gate.

Figure 6. Karnaugh map that illustrate alternative ESOP syntheses of function F_2 : a) for expression $ef \oplus c'd'ef \oplus a'b'c'def$ and b) for expression $ef \oplus c'ef \oplus c'def \oplus a'b'c'def$.

Based on the above examples, the threshold value for the fractional covering criterion occurs at the point when the AND gate with the smallest number of inputs performs an asserting role and all subsequent AND gates, each of which covers one-fourth the number of minterms as compared to the prior gate. Under these conditions the total number of asserted minterms is $m = 2^k - (2^{k-2} + 2^{k-4} + \dots + 2^0)$ for some even integer k . Based on this threshold, expressions must satisfy a $\lim_{k \rightarrow \infty} \left(\frac{m}{2^k} \right) = \frac{2}{3}$ covering criterion in order to be acceptable. Observe that these criteria used in my synthesis algorithms essentially reduce the search and thus make my methods to be “heuristic optimization” rather than exhaustive search, although tree searches are used in them.

Discussion:

As we can see in section 2.5.2 the 2/3 criterion is giving better solution than the 3/4 criterion. But regardless of what the values are, 1/2, 2/3, 3/4, etc. this is just a HEURISTIC method. A heuristic method never guarantees that a global minimization can ALWAYS be obtained. You can always find a few examples/functions to claim that a value is better than others. Based on the examination of 35 different types of benchmark functions in Section 3.1.4, the “2/3 criterion” gives the best solution for most of them (33/35). So, this “2/3 criterion” will be used in this dissertation. The problem of best value of the heuristic parameter could also be solved by setting this criterion as a parameter and randomize it for each run to check for the minimum solution.

Chapter 3: SYNTHESIS OF EPOE CIRCUITS FOR COMPLETELY SPECIFIED SINGLE OUTPUT BOOLEAN FUNCTIONS

To achieve the first objective “*examine the methods that will convert a single output Boolean function of N input variables $(a,b,c,d, \dots)$ into EPOE expression*”, two methods are presented in this dissertation:

- 1) **template matching method** based on the library of POE templates that is created before synthesis, and
- 2) **algebraic approach** based on a special type of factorization.

3.1 Template matching method based on the library of templates:

There are two variants of the EPOEM-1 algorithm called EPOEM-1s and EPOEM-1f. Both algorithms convert a Boolean function of N input variables $(a,b,c,d, \dots)$ into EPOE form using a template-matching method. The templates are specific ordered sets of minterms that can be used in EPOE synthesis as in Table 2. Every template represents either an EXOR-sum of literals or a POE. The algorithms use a strategy of searching for a template T_i which intersects the function's ON-set in over *two-thirds* of the template's minterms $M(T_i)$, more formally $|M(T_i) \cap ON| > \frac{2}{3}|M(T_i)|$ (8). Here $|f|$ denotes the number of minterms in function f . The best matching template is the template that satisfies (8) and EXORed with the function to produce a remainder function with the smallest ON-set. The procedure of selecting the best matching template that satisfies (8) is iteratively applied until the remainder function becomes an empty set of minterms (which means the remainder function becomes 0). Both algorithms require a library of POE templates, calculated in

advance and grouped together by the number of product terms employed. The generation of two template libraries, one for each algorithm, is presented in Section 3.1.1.

3.1.1 Generation of Template Libraries for N-variable Functions:

This work introduces two template libraries, one, called a single expression library [14], contains only one minimum cost POE expression for each template, and the other, called a full expression library, contains all possible POE expressions for each template. Because the single expression library is smaller than the full expression library for a certain number of inputs, it allows synthesis of functions with more inputs than the full expression library (8 compare to 6) which is constrained by disk memory size. With the full expression library, searches can be performed for POE expressions that have common EXOR-sums. Common EXOR-sums are factored in order to reduce the number of inputs of Toffoli gates, thus reducing the quantum cost. The individual library templates are grouped together by the number of minterms that they cover, referred to as a *level*, which satisfies the equation: $|M(T_i)| = 2^{N-level}$. Both libraries share the same level 0 and level 1 templates, but differ in their level 2 and higher templates. This is shown in Table 2. The template's organization by levels is as follows:

- i) Level 0 contains only one template (*set1*), which covers all the minterms in the truth table.
- ii) Level 1 contains $2^{N+1} - 2$ templates, which are all the affine functions of N -variables that cover $\frac{1}{2}$ of the minterms in the truth table, and their associated template expressions.

- iii) Level 2 contains all the templates that cover 2^{N-2} minterms in the truth table. The single expression library contains a single template expression for each template, and the full library contains all template expressions for each template.

...

- iv) Level n-1 contains all the templates that cover exactly two minterms in the truth table and their associated template expressions.

The algorithms for Algorithm-1a, which is for the single expression library generation, and Algorithm-1b, which is for the full library generation, follow.

Algorithm 1a: Generates the single expression library from level m ($m > 1$) through n-1:

```

n := number of variables in the given function
m := 2      //level of the library currently being generated
Load level(1) of the library  // contains all the affine functions
while m < n:
    CTP := Cartesian product of level(1) and level(m - 1) of the
           library
    For each (mem1, mem2) pair in CTP do:
        intersect := product (mem1, mem2)
        If intersect is not empty
            newname := concatenate(expression(mem1),
                                   expression(mem2))
            If intersect already exists in level(m) of the
                                   library
                Current := level(m).find(intersect).expression
                Compare the cost of newname vs. Current
                Store the expression with the minimum cost
            Else, add the pair (intersect, newname) to the
                                   library
    m = m + 1

```

Algorithm 1b: Generate the full library from level m ($m > 1$) through n-1:

```

n := number of variables in the given function
m := 2      //level of the library currently being generated
Load level(1) of the library  // contains all the affine functions
while m < n:
    CTP := Cartesian product of level(1) and level(m - 1) of the
           library
    For each (mem1, mem2) pair in CTP do:
        intersect := product (mem1, mem2)
        If intersect is not empty

```

```

newname := concatenate (expression(mem1),
                        expression(mem2))
Add the pair (intersect, newname) to library
m = m + 1

```

The time complexity of algorithm 1a and 1b which use to generate the template library is $O(N^2/2)$, where N is a number of templates in level $(m-1)$, because it creates a set of nonduplicate Cartesian products.

Table 2. Examples of POE template expressions for functions of three variables.

Level	Minterms Covered (Template)	POE Template expressions	
		Single expression template library	Full expressions template library
0	{000,001,010,011,100,101,110,111}	1	1
1	{100,101,110,111}	a	a
1	{010,011,100,101}	$a \oplus b$	$a \oplus b$
1	{001,010,100,111}	$a \oplus b \oplus c$	$a \oplus b \oplus c$
2	{100,101}	$a(a \oplus b)$	$a(a \oplus b); a(b \oplus 1); (a \oplus b)(b \oplus 1)$
2	{100,011}	$(a \oplus c)(a \oplus b)$	$(a \oplus c)(a \oplus b); (a \oplus c)(b \oplus c \oplus 1); (a \oplus b)(b \oplus c \oplus 1)$

3.1.2 EPOEM-1s using single expression template library:

The EPOEM-1s Algorithm which uses the single expression template library is shown below. An example of EPOEM-1s synthesis is presented subsequently.

A. EPOEM-1s Algorithm:

```

INPUT: ON_set of n-variable function
OUTPUT: EPOE expressions for given function of n variables

k := 2/3
remainder := input(ON_set)    // input function to be synthesized
n := number of input variables
level := 0                    // level in the template library
result := NIL // a string that accumulates the result expressions

// Check if full map match:
If remainder.length() > k.power(2, n)
    result := "1"    //NOT gate
    remainder := remainder EXOR set1
    level := level + 1

```

```

While remainder.length() > 1
  If remainder.length() <= k·power(2, n - level)
    level := level + 1
  Else
    For all templates in library at current level:
      Find all templates where the intersection of the template and the
      remainder covers at least 2/3 of the template
    If such templates exist:
      Choose the template that has the largest intersection with the
      remainder, in the case of tie select randomly among the
      best
      remainder := remainder EXOR template
      result = concatenate (result, " $\oplus$ ", template_expression)
    Else
      level: = level + 1

If remainder.length() > 0
  // In this case remainder is a single minterm which is a product
  // of degenerate EXOR-sums
  result = concatenate (result, " $\oplus$ ", remainder.text())

Return result

```

The time complexity for algorithm EPOEM-1s is $O(N)$, with N is the number of templates in the library, because it searches the hash table library.

Figure 7. Karnaugh maps that illustrate template selection for EPOEM-1s and EPOEM-1f.

Figure 7 illustrates one iteration of the above algorithm in which the best partially intersecting POE template is selected for function $F_3(a, b, c, d) = \sum(1,2,3,5,7,8,9,10,14)$ (here true minterms are specified as natural numbers). Template $(a \oplus d)$ intersects with seven minterms in F_3 . Template $(b \oplus 1)$ intersects with six minterms in F_3 . In searching

through the remaining templates, no other templates were found to intersect with more than seven minterms, so template $(a \oplus d)$ is selected as the result.

B. Example 1

Example 1 illustrates a complete EPOE synthesis of a fully specified function. Below a complete EPOEM-1s synthesis will be performed on the function $F_4(a, b, c, d) = \sum(1,2,3,5,7,8,9,10)$. Assume that the four-variable POE single expression template library has been calculated in advance.

- The *remainder* is initialized to the ON-set = {0001, 0010, 0011, 0101, 0111, 1000, 1001, 1010}.
- At *level* = 0 the *remainder* has only 8 minterms, and since $8 < (2/3)2^4$ the function is not negated. Consequently a “1” is not appended to the *result*.
- At *level* = 1 a POE template library search is performed and the templates $(a \oplus d)$ and $(b \oplus 1)$ are both found to intersect with 6 minterms in the *remainder* which is shown in Figure 8. Since $6 > (2/3)2^3$ both templates are acceptable, therefore $(a \oplus d)$ is randomly selected. The selected template is EXORed with the *remainder* and its expression is appended to the *result*.
- The new *remainder* becomes {0010, 1001, 1100, 1110}, which has only 4 minterms ($4 < (2/3)2^3$) so go to next *level*
- At *level* = 2 a POE template library search is performed and many acceptable templates are found with equal quantum cost with 3 intersecting minterms. The template $(a \oplus b \oplus 1)(d \oplus 1)$ is randomly selected and is shown in Figure 9. The

selected template is EXORed with the *remainder* and its expression is appended to the *result*.

- The new *remainder* becomes $\{0000, 1001\}$, which has only 2 minterms ($2 < (2/3)2^2$) so go to next *level*
- At *level* = 3 a POE template library search is performed and the only acceptable template found is $(b \oplus 1)(c \oplus 1)(a \oplus d \oplus 1)$, which intersects the remaining two minterms. The selected template is EXORed with the *remainder* and its expression is appended to the *result*.
- The new *remainder* becomes $\{\}$ which completes the synthesis. The final value of the *result* is as follows:

$$F_4(a, b, c, d) = (a \oplus d) \oplus (a \oplus b \oplus 1)(d \oplus 1) \oplus (b \oplus 1)(c \oplus 1)(a \oplus d \oplus 1)$$

Figure 8. Karnaugh maps that illustrate the remainder functions generated from function F_4 for two templates in EPOEM-1s

Figure 9. Karnaugh maps that illustrate the remainder function evolution in EPOEM-1s

There are many quantum circuit realizations for any expressions of a function. Example function F_4 can be realized in one of the two ways shown in Figure 10. This observation is a fundament of our randomization approaches to algorithms creation. A circuit with no ancilla lines but an output line using mirrors to restore the values of input lines is shown in Figure 10a. An alternative circuit with unlimited ancilla lines (garbage lines) is shown in Figure 10b, with reduced pulse related quantum cost but increased line related quantum cost [8]. The realization in Figure 10b has the additional benefit that EXOR-sums may be reused as inputs to additional POE expressions (not shown here). Another benefit of this type of realization is the ability to share sub-functions in multiple output circuits.

Figure 10. Two realizations of the EPOE circuits for expression $F_4(a,b,c,d) = (a \oplus d) \oplus (a \oplus b \oplus 1)(d \oplus 1) \oplus (b \oplus 1)(c \oplus 1)(a \oplus d \oplus 1)$. a) Circuit with no garbage line but with mirror gates b) Circuit with no mirror gates but with garbage lines.

Compare with Exorcism-4 output, $F_4 = b'cd' \oplus a'd \oplus ab'c$ and its circuit realization that has quantum cost of 39 (see Figure 11), it can be appreciated that EPOEM-1s result has a cost reduced by 10.2% with respect to the Exorcism-4 result. Exorcism-4 has a higher cost because of using multi-input Toffoli gates.

Figure 11. Standard ESOP-like circuit realization of function $F_4(a, b, c, d) = b'cd' \oplus a'd \oplus ab'c$ with only Toffoli and NOT gates. The quantum cost is 39.

3.1.3 EPOEM-1f using full expression template library:

The EPOEM-1f Algorithm, which uses the full expression template library, is shown below. An example of EPOEM-1f synthesis is presented subsequently

A. EPOEM-1f Algorithm

- ...
 - Same as EPOE-1s Algorithm until result is returned.
 - ...
 - Create all of the possible result expressions by choosing every combination of template expressions from each template's equivalent expression list. Each template at level higher than 2 has multiple expressions.
 - Find all the common POE terms for each result expression
 - Factorize each of the common POE terms out. There can be many different ways to factorize the expression, all of which will be considered.
 - Calculate the cost of each expression
 - The expression with minimum cost is the final result
-

The time complexity for searching the hash table library is $O(N_1)$ with N_1 is the number of templates in the library, for creating all combination is $O(N_2^2)$ with N_2 is the number of alternative expressions for each template, for finding common POE is $O(N_3)$ with N_3 is the number of common POE and for factorization based on hash table implementation is $O(1)$. So the overall time complexity of algorithm EPOEM-1f is $O(N^2)$ with N is the number of alternative expressions for each template

B. Example 2:

Below a complete EPOEM-1f synthesis will be performed on the function $F_4(a, b, c, d) = \sum(1,2,3,5,7,8,9,10)$. Assume that the four-variable POE full expression template library has been calculated in advance.

- Apply the method as in Example 1. The resulting EPOE expression is as follow:

$$F_4 = \left\{ \begin{array}{l} (b \oplus 1) \oplus \\ [(a \oplus 1)(b \oplus d \oplus 1); (a \oplus 1)(a \oplus b \oplus d \oplus 1); (b \oplus d \oplus 1)(a \oplus b \oplus d \oplus 1)] \oplus \\ [(b \oplus 1)(a \oplus d \oplus 1)(a \oplus b \oplus c \oplus d); (a \oplus d \oplus 1)(a \oplus c \oplus d)(a \oplus b \oplus d \oplus 1); \\ (b \oplus 1)(a \oplus c \oplus d)(a \oplus b \oplus d \oplus 1); \dots \end{array} \right\}$$

Above, a “;” is used to separate each alternative expression of the template. These expressions will be used to determine the minimum quantum cost.

- Expand result to generate all possible EPOE expressions of F_4 . This is performed by selecting every combination of multiple alternative POE expressions, e.g.:

$$F_4 = (b \oplus 1) \oplus (a \oplus 1)(b \oplus d \oplus 1) \oplus (b \oplus 1)(a \oplus d \oplus 1)(a \oplus b \oplus c \oplus d) \quad (9)$$

$$F_4 = (b \oplus 1) \oplus (b \oplus d \oplus 1)(a \oplus b \oplus d \oplus 1) \oplus (b \oplus 1)(a \oplus c \oplus d)(a \oplus b \oplus d \oplus 1) \quad (10)$$

$$F_4 = (b \oplus 1) \oplus (a \oplus 1)(a \oplus b \oplus d \oplus 1) \oplus (b \oplus 1)(a \oplus c \oplus d)(a \oplus b \oplus d \oplus 1) \quad (11)$$

...

- Perform factorization on all EPOE expressions with common POE terms. There can be multiple ways to factorize even a single EPOE expression, e.g. (10). Calculate the quantum cost of each result, using a circuit realization model where inputs are mirrored back to their original values and one additional ancilla line is required for each common POE term:

$$(9) \Rightarrow F_4 = (b \oplus 1) [(a \oplus d \oplus 1)(a \oplus b \oplus c \oplus d) \oplus 1] \oplus (a \oplus 1)(b \oplus d \oplus 1) \quad \text{Cost} = 35$$

$$(10) \Rightarrow \begin{cases} F_4 = (b \oplus 1) [(a \oplus c \oplus d)(a \oplus b \oplus d \oplus 1) \oplus 1] \oplus (b \oplus d \oplus 1)(a \oplus b \oplus d \oplus 1) & \text{Cost} = 39 \\ F_4 = (b \oplus 1) \oplus (a \oplus b \oplus d \oplus 1) [(b \oplus d \oplus 1) \oplus (b \oplus 1)(a \oplus c \oplus d)] & \text{Cost} = 31 \end{cases}$$

$$(11) \Rightarrow \begin{cases} F_4 = (b \oplus 1) [(a \oplus c \oplus d)(a \oplus b \oplus d \oplus 1) \oplus 1] \oplus (a \oplus 1)(a \oplus b \oplus d \oplus 1) & \text{Cost} = 37 \\ F_4 = (b \oplus 1) \oplus (a \oplus b \oplus d \oplus 1) [(a \oplus 1) \oplus (b \oplus 1)(a \oplus c \oplus d)] & \text{Cost} = 28 \end{cases}$$

...

- Compare all the costs and select the EPOE expression with the minimum quantum cost.

The lowest cost is 28 which leads to the selection of the following EPOE expression as the algorithm result:

$$F_4 = (b \oplus 1) \oplus (a \oplus b \oplus d \oplus 1) [(a \oplus 1) \oplus (b \oplus 1)(a \oplus c \oplus d)]$$

The circuit realization for this expression of function F_4 is shown in Figure 12.

Figure 12. Circuit for function F_4 produced by EPOEM-1f algorithm

3.1.4 Experimental Results:

EPOEM programs have been implemented in Python and tested extensively on Unix and Windows workstations. The experimental results below have been received on a 2.9 GHz Intel Core i7 PC under Microsoft Windows 8.1.

To verify and compare EPOEM-1s and EPOEM-1f algorithms, several single-output benchmark functions were taken from Revlib's page [17], Maslov's page [18] and [14] for synthesis testing. A comparison of the results from EPOEM-1s, EPOEM-1f, EXORCISM-4 [9] and Revlib (if given) are shown in Table 3 for four and five variable functions. The respective comparisons of the results from EPOEM-1s and Exorcism-4 [9] are shown in Table 4 for six or more variable functions. The best result for each benchmark is shown in bold font. Compared with EXORCISM-4 over many benchmark functions, both EPOEM-

1s and EPOEM-1f consistently produced solutions of equal or lower quantum cost with improvements ranging up to typically 50%, and in some cases up to 85%. EPOEM-1f always gives the better solution than EPOEM-1s but will cost additional ancilla line for each common POE term. In addition to the above evaluation method, I have also done statistical analysis based on randomization, and I tested my algorithm on some random benchmark functions as shown in Figure 13.

Table 3. Four and five input variable functions synthesized with EXORCISM-4 vs. EPOEM vs. REVLIB

Function	Results		Quantum Cost
lt41	EXORCISM-4	$ab'd' \oplus b'c'd \oplus a'd$	41
	EPOEM-1s	$((a \oplus d)(a \oplus b)) \oplus ((a \oplus c)(b \oplus 1)(d))$	27
	EPOEM-1f	$((a \oplus d)(a \oplus b)) \oplus ((a \oplus c)(b \oplus 1)(d))$	27
lt42	EXORCISM-4	$ac \oplus a'b' \oplus d' \oplus a'bcd'$	50
	EPOEM-1s	$(c \oplus d \oplus 1) \oplus ((a \oplus 1)(b \oplus c \oplus 1)) \oplus ((a \oplus 1)(b)(c)(d \oplus 1))$	49
	EPOEM-1f	$(c \oplus d \oplus 1) \oplus (a \oplus 1) [(b \oplus c \oplus 1) \oplus ((b)(c)(d \oplus 1))]$	32 (+1 anc. line)
lt43	EXORCISM-4	$b'cd' \oplus a'd \oplus ab'c'$	41
	EPOEM-1s	$(b \oplus 1) \oplus ((a \oplus 1)(b \oplus d \oplus 1)) \oplus ((b \oplus 1)(a \oplus d \oplus 1)(c))$	33
	EPOEM-1f	$(b \oplus 1) \oplus (a \oplus b \oplus d \oplus 1) [(a \oplus 1) \oplus ((a \oplus d \oplus 1)(a \oplus c \oplus 1))]$	30 (+1 anc. line)
lt44	EXORCISM-4	$b'c'd' \oplus a'b'c' \oplus ab'd' \oplus a'$	58
	EPOEM-1s	$(a \oplus 1) \oplus ((b \oplus 1)(c \oplus 1)) \oplus ((c \oplus d)(b \oplus 1)(a \oplus d))$	31
	EPOEM-1f	$(a \oplus 1) \oplus ((b \oplus 1) [(c \oplus 1) \oplus ((c \oplus d)(a \oplus d))])$	22 (+1 anc. line)
lt45	EXORCISM-4	$1 \oplus acd \oplus abd \oplus a'b'c'd'$	64
	EPOEM-1s	$1 \oplus ((a \oplus d \oplus 1)(a \oplus b \oplus c \oplus 1)) \oplus ((a \oplus 1)(b)(c)(d \oplus 1))$	49
	EPOEM-1f	$1 \oplus ((a \oplus d \oplus 1)(a \oplus b \oplus c \oplus 1)) \oplus ((a \oplus 1)(b)(c)(d \oplus 1))$	49
4gt4_20	EXORCISM-4	$1 \oplus a'b' \oplus a'bc'd'$	45
	EPOEM-1s	$1 \oplus (a \oplus 1)(b \oplus 1) \oplus (a \oplus 1)b(c \oplus 1)(d \oplus 1)$	45
	EPOEM-1f	$1 \oplus (a \oplus 1) [(b \oplus 1) \oplus b(c \oplus 1)(d \oplus 1)]$	28 (+1 anc. line)
	RevLib		54
4gt5_21	EXORCISM-4	$a \oplus a'bc$	16
	EPOEM-1s	$a \oplus (a \oplus 1)bc$	16
	EPOEM-1f	$a \oplus (a \oplus 1)bc$	16
	RevLib		21
4gt10_22	EXORCISM-4	$ab \oplus ab'cd$	36
	EPOEM-1s	$ab \oplus a(b \oplus 1)cd$	36
	EPOEM-1f	$a [b \oplus (b \oplus 1)cd]$	21 (+1 anc. line)
	RevLib		34
4gt12_24	EXORCISM-4	$abc \oplus abc'd$	44
	EPOEM-1s	$ab \oplus ab(c \oplus 1)(d \oplus 1)$	38
	EPOEM-1f	$ab [1 \oplus (c \oplus 1)(d \oplus 1)]$	23 (+1 anc. line)
	RevLib		41
4mod5_8	EXORCISM-4	$a'd \oplus a'b' \oplus cd \oplus b'c$	28
	EPOEM-1s	$(a \oplus c \oplus 1) (b \oplus d \oplus 1)$	13

	EPOEM-1f	$(a \oplus c \oplus 1)(b \oplus d \oplus 1)$	13
	RevLib		
4sf_232	EXORCISM-4	$c \oplus b'c \oplus ab'c \oplus a'bd$	38
	EPOEM-1s	$(a \oplus b \oplus c \oplus d) \oplus (c \oplus 1)d \oplus ab(c \oplus d \oplus 1)$	31
	EPOEM-1f	$(a \oplus b \oplus c \oplus d) \oplus d[(c \oplus 1) \oplus (a \oplus b \oplus 1)(b \oplus c \oplus d)]$	28 (+1 anc. line)
	RevLib		51
lt51	EXORCISM-4	$c' \oplus b'e \oplus c'de \oplus abce \oplus a'b'c'e' \oplus a'bc'd'e$	142
	EPOEM-1s	$(c \oplus e \oplus 1) \oplus ((a \oplus 1)(b \oplus c \oplus 1)(b \oplus e \oplus 1)) \oplus ((c \oplus 1)(b \oplus d)d) \oplus (a \oplus 1)b(c \oplus 1)(d \oplus 1)e$	112
	EPOEM-1f	$(c \oplus e \oplus 1)[1 \oplus (a \oplus 1)(b \oplus e \oplus 1)] \oplus (c \oplus 1)[(b \oplus c \oplus d)(b \oplus d \oplus e \oplus 1)] \oplus (a \oplus 1)b(d \oplus 1)e]$	76 (+2 anc. line)
lt52	EXORCISM-4	$e \oplus bc' \oplus bc'd'e \oplus abcd'e'$	104
	EPOEM-1s	$e \oplus b(c \oplus 1) \oplus b(d \oplus 1)(c \oplus e) \oplus (a \oplus 1)bc(d \oplus 1)(e \oplus 1)$	92
	EPOEM-1f	$e \oplus (b \oplus c)(c \oplus 1) \oplus b(d \oplus 1)[(c \oplus e) \oplus (a \oplus 1)c(e \oplus 1)]$	45 (+1 anc. line)
5alu_9	EXORCISM-4	$a'e' \oplus a'd' \oplus c'd'e' \oplus b'de$	52
	EPOEM-1s	$(a \oplus 1) \oplus (a \oplus b \oplus d \oplus d)(c \oplus d \oplus 1) \oplus (b \oplus c \oplus 1)(d \oplus 1)(e \oplus 1)$	39
	EPOEM-1f	$(a \oplus 1) \oplus (d \oplus e \oplus 1)[(a \oplus b \oplus d \oplus 1) \oplus (b \oplus c \oplus 1)(b \oplus c \oplus e \oplus 1)]$	34 (+1 anc. line)
	RevLib		
5ex2_151	EXORCISM-4	$a' \oplus b'e \oplus b'c' \oplus cde \oplus a'bc'e' \oplus ac'd'e' \oplus a'b'cde$	143
	EPOEM-1s	$(a \oplus c \oplus e) \oplus a(b \oplus c \oplus d \oplus 1)(c \oplus e \oplus 1) \oplus (a \oplus 1)bc(d \oplus 1)e$	93
	EPOEM-1f	$(a \oplus c \oplus e)[1 \oplus (b \oplus d \oplus e \oplus 1)(c \oplus e \oplus 1)] \oplus (a \oplus 1)bc(d \oplus 1)e$	90 (+1 anc. line)
	Revlib		141
5ex3_152	EXORCISM-4	$ab \oplus bc'e \oplus abcd \oplus a'c'd'e$	84
	EPOEM-1s	$(a \oplus b \oplus 1)(a \oplus c \oplus 1)(d \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)(e \oplus 1) \oplus (a \oplus 1)(c \oplus 1)(b \oplus e \oplus 1)d$	81
	EPOEM-1f	$(a \oplus b \oplus 1)(a \oplus c \oplus 1)(d \oplus 1) \oplus (a \oplus b \oplus e \oplus 1)(c \oplus 1)[(a \oplus b \oplus c \oplus 1) \oplus (b \oplus e \oplus 1)(a \oplus b \oplus d \oplus e)]$	66 (+1 anc. line)
	Revlib		79
5majority_176	EXORCISM-4	$d \oplus abd' \oplus a'cd'e \oplus b'cd'e \oplus abc'd'e'$	149
	EPOEM-1s	$c \oplus (a \oplus b \oplus c \oplus 1)(d \oplus 1)e \oplus (a \oplus b \oplus 1)(a \oplus c \oplus 1)(d \oplus 1) \oplus a'b'c'd'e'$	116
	EPOEM-1f	$c \oplus (d \oplus 1)[(a \oplus b \oplus 1)(b \oplus d) \oplus (a \oplus b \oplus c \oplus 1)(a \oplus b \oplus e \oplus 1) \oplus a'b'c'e']$	73 (+1 anc. line)
	Revlib		126
5rd53f1	EXORCISM-4	$ade \oplus bcd \oplus abce \oplus ab'c'de \oplus a'bcde'$	177
	EPOEM-1s	$(a \oplus b \oplus c \oplus 1)de \oplus (a \oplus b \oplus 1)(a \oplus c \oplus 1)(a \oplus d \oplus e \oplus 1) \oplus (a \oplus b \oplus 1)(a \oplus c \oplus 1)(a \oplus d \oplus 1)(a \oplus e \oplus 1)$	91
	EPOEM-1f	$(a \oplus b \oplus 1)(a \oplus c \oplus 1)(a \oplus b \oplus c) \oplus (a \oplus b \oplus c \oplus d)(d \oplus e \oplus 1)[(a \oplus b \oplus c \oplus 1) \oplus (b \oplus c \oplus d \oplus e \oplus 1)(b \oplus 1)]$	70 (+1 anc. line)
5rd53f2	EXORCISM-4	$ab \oplus a'd \oplus b'e \oplus de \oplus b'c'd \oplus a'c'e \oplus ace' \oplus bcd'$	88
	EPOEM-1s	$(a \oplus b \oplus c \oplus d \oplus 1) \oplus (a \oplus b \oplus 1)(a \oplus c \oplus 1) \oplus (d \oplus e \oplus 1)(a \oplus b \oplus c \oplus d)$	37
	EPOEM-1f	$(a \oplus b \oplus c \oplus d \oplus 1) \oplus (a \oplus b \oplus 1)(a \oplus c \oplus 1) \oplus (d \oplus e \oplus 1)(a \oplus b \oplus c \oplus d)$	37

Table 4. Six and more input variable functions synthesized with EXORCISM-4 vs. EPOEM-1s

Function	Quantum Cost		Function	Quantum Cost	
	EPOEM-1s	EXORCISM-4		EPOEM-1s	EXORCISM-4
lt61	187	293	8newill	684	1239
sym6_63	136	857	8newtag	483	683
7con1f1	119	141	8rd84f1	60	277
7con2f2	60	68	8rd84f2	8	24
7rd73f1	50	211	8rd84f3	509	509
7rd73f2	7	19	8rd84f4	333	4824
7rd73f3	203	1337

a) Function 4sf with best quantum cost of 27

b) Function 5ex3 with best quantum cost of 69

c) Function 5majority with best quantum cost of 112

d) Function 6sym with best quantum cost of 136

Figure 13. Frequency distribution of quantum cost results from 50 random runs using EPOEM-1s to synthesize functions: a) 4sf, b) 5ex3, c) 5majority, d) 6sym

3.2 Algebraic method based on factorization:

Algorithm EPOEM-2 is a purely algebraic approach based on a special type of factorization. This factorization method can be expressed by the following two propositions:

3.2.1 Proposition 1:

The EXOR-sum of any two POE terms of the same support family can be factorized into a single POE term; moreover there exists usually more than one such factorization.

A. Examples to illustrate Proposition 1:

- Applying Proposition 1 to the two minterm function $F_5 = a'b'c'd'e \oplus abcd'e$, which yields multiple factorizations such as :

$$\begin{aligned} F_5 &= e(a \oplus b \oplus 1)(a \oplus c \oplus 1)(a \oplus d) \\ &\quad \text{or} \\ F_5 &= e(a \oplus b \oplus 1)(b \oplus c \oplus 1)(c \oplus d) \\ &\quad \text{or} \\ &\quad \dots \end{aligned}$$

- Applying Proposition 1 to the two POE term function $F_6 = (a \oplus c)(a \oplus b \oplus 1)(c \oplus d) \oplus (a \oplus c)(a \oplus b)(c \oplus d \oplus 1)$, which yields only one factorization:

$$F_6 = (a \oplus c)(a \oplus b \oplus c \oplus d)$$

Before we are going to prove the Proposition 1, let's observe this Lemma 1

B. Lemma 1:

For any integer $n \geq 0$ and let c_{Ai}, c_{Bi} ($1 \leq i \leq n$) denote any set of constants that satisfy the condition $c_{Ai} = c_{Bi} \oplus 1$ for each i . Then for any unknowns x_i , ($1 \leq i \leq n$) the following relationship holds:

$$\prod_{i=1}^n (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^n (x_i \oplus c_{B_i}) = \prod_{i=2}^n (x_1 \oplus x_i \oplus c_{A_1} \oplus c_{A_i} \oplus 1)$$

Proof:

Lemma 1 is true for $n = 1, 2$, and 3 , as is easily verified below.

- $n = 1$:

$$\prod_{i=1}^1 (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^1 (x_i \oplus c_{B_i}) = (x_1 \oplus c_{A_1}) \oplus (x_1 \oplus c_{B_1}) = (x_1 \oplus x_1 \oplus c_{A_1} \oplus c_{A_1} \oplus 1)$$

- $n = 2$:

$$\begin{aligned} \prod_{i=1}^2 (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^2 (x_i \oplus c_{B_i}) &= (x_1 \oplus c_{A_1})(x_2 \oplus c_{A_2}) \oplus (x_1 \oplus c_{B_1})(x_2 \oplus c_{B_2}) \\ &= (x_1 \oplus x_2 \oplus c_{A_1} \oplus c_{A_2} \oplus 1) \end{aligned}$$

- $n = 3$:

$$\begin{aligned} \prod_{i=1}^3 (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^3 (x_i \oplus c_{B_i}) &= (x_1 \oplus c_{A_1})(x_2 \oplus c_{A_2})(x_3 \oplus c_{A_3}) \oplus (x_1 \oplus c_{B_1})(x_2 \oplus c_{B_2})(x_3 \oplus c_{B_3}) \\ &= (x_1 \oplus x_2 \oplus c_{A_1} \oplus c_{A_2} \oplus 1)(x_1 \oplus x_3 \oplus c_{A_1} \oplus c_{A_3} \oplus 1) \end{aligned}$$

We proceed by induction on n . Suppose that Lemma 1 is true for $n = k$, for some $k \geq 3$.

Then, for this value of k , we are assuming that

$$\prod_{i=1}^k (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^k (x_i \oplus c_{B_i}) = \prod_{i=2}^k (x_1 \oplus x_i \oplus c_{A_1} \oplus c_{A_i} \oplus 1) \quad (1)$$

Now we wish to prove that Lemma 1 is true for $n = k + 1$, which means:

$$\prod_{i=1}^{k+1} (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^{k+1} (x_i \oplus c_{B_i}) = \prod_{i=2}^{k+1} (x_1 \oplus x_i \oplus c_{A_1} \oplus c_{A_i} \oplus 1).$$

To establish the above equation, we begin with the right-hand side:

$$\begin{aligned}
RHS &= \prod_{i=2}^{k+1} (x_1 \oplus x_i \oplus c_{A_1} \oplus c_{A_i} \oplus 1) = \left[\prod_{i=2}^k (x_1 \oplus x_i \oplus c_{A_1} \oplus c_{A_i} \oplus 1) \right] (x_1 \oplus x_{k+1} \oplus c_{A_1} \oplus c_{A_{k+1}} \oplus 1) \\
&= \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^k (x_i \oplus c_{B_i}) \right] (x_1 \oplus x_{k+1} \oplus c_{A_1} \oplus c_{A_{k+1}} \oplus 1) \quad (\text{by using (1)}) \\
&= \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \right] (x_1 \oplus x_{k+1} \oplus c_{A_1} \oplus c_{A_{k+1}} \oplus 1) \oplus \left[\prod_{i=1}^k (x_i \oplus c_{B_i}) \right] (x_1 \oplus x_{k+1} \oplus c_{A_1} \oplus c_{A_{k+1}} \oplus 1) \\
&= \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \right] (x_1 \oplus c_{A_1} \oplus 1) \oplus \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \right] (x_{k+1} \oplus c_{A_{k+1}} \oplus 1) \oplus \\
&\quad \left[\prod_{i=1}^k (x_i \oplus c_{B_i}) \right] (x_1 \oplus c_{A_1}) \oplus \left[\prod_{i=1}^k (x_i \oplus c_{B_i}) \right] (x_k \oplus c_{A_k} \oplus 1) \\
&= 0 \oplus \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \right] (x_{k+1} \oplus c_{A_{k+1}} \oplus 1) \oplus 0 \oplus \left[\prod_{i=1}^k (x_i \oplus c_{B_i}) \right] (x_k \oplus c_{A_k} \oplus 1) \\
&\quad (\text{because: } (x_1 \oplus c_{A_1})(x_1 \oplus c_{A_1} \oplus 1) = (x_1 \oplus c_{B_1})(x_1 \oplus c_{B_1} \oplus 1) = 0 \text{ and } (x_1 \oplus c_{A_1}) = (x_1 \oplus c_{B_1} \oplus 1)) \\
&= \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \right] (x_{k+1} \oplus c_{A_{k+1}} \oplus 1) \oplus \left[\prod_{i=1}^k (x_i \oplus c_{B_i}) \right] (x_k \oplus c_{A_k} \oplus 1) = LHS
\end{aligned}$$

So by induction, Lemma 1 is proved, and we can now use Lemma 1 to prove

Proposition 1.

C. Proof of Proposition 1:

Let the two given POE terms be

$$A = (x_1 \oplus c_{A_1})(x_2 \oplus c_{A_2}) \dots (x_n \oplus c_{A_n}),$$

$$B = (x_1 \oplus c_{B_1})(x_2 \oplus c_{B_2}) \dots (x_n \oplus c_{B_n})$$

where each coefficient c_{A_i} and c_{B_i} takes one of the two values: 0 or 1

Suppose A and B have exactly k different coefficients $c_{A_i} \neq c_{B_i}$. Without loss of generality, we may assume $c_{A_i} \neq c_{B_i}$ for $1 \leq i \leq k$, and $c_{A_i} = c_{B_i}$ for $k + 1 \leq i \leq n$.

Then

$$\begin{aligned}
A \oplus B &= \prod_{i=1}^n (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^n (x_i \oplus c_{B_i}) \\
&= \prod_{i=k+1}^n (x_i \oplus c_{A_i}) \left[\prod_{i=1}^k (x_i \oplus c_{A_i}) \oplus \prod_{i=1}^k (x_i \oplus c_{B_i}) \right] \\
&= \prod_{i=k+1}^n (x_i \oplus c_{A_i}) \left[\prod_{i=2}^k (x_1 \oplus x_i \oplus c_{A_1} \oplus c_{A_i} \oplus 1) \right] \quad (\text{by using Lemma 1})
\end{aligned}$$

So, Proposition 1 is proved.

As we can see, Proposition 1 factors two POE terms with N EXOR-sums into one POE term with $N - 1$ EXOR-sums, which means Proposition 1 replaces the circuit with two $N \times N$ Toffoli gates with the circuit with one $(N - 1) \cdot (N - 1)$ Toffoli gate, so the quantum cost reduces from $2 \cdot (2^N - 3)$ to $(2^{N-1} - 3)$.

3.2.2 Proposition 2:

The EXOR-sum of any three POE terms of the same support family can be factorized into a pair of two POE terms; moreover there is usually more than one factorization.

A. Examples to illustrate Proposition 2:

Figure 14. Karnaugh maps that illustrate applications of Proposition 2

B. Proof of Proposition 2:

Let the three given POE terms be $A = (x_1 \oplus c_{A1})(x_2 \oplus c_{A2}) \dots (x_n \oplus c_{An})$, $B = (x_1 \oplus c_{B1})(x_2 \oplus c_{B2}) \dots (x_n \oplus c_{Bn})$ and $C = (x_1 \oplus c_{C1})(x_2 \oplus c_{C2}) \dots (x_n \oplus c_{Cn})$

Each coefficient c_{Ai} , c_{Bi} and c_{Ci} can take one of the two values: 0 or 1

In order to prove Proposition 2 let us first observe that the completing POE term is a POE term that can be factorized together with the three given POE terms.

Our objective is to find the completing POE term. This task can be achieved as follows:

1. Let $A = (x_1 \oplus c_{A1})(x_2 \oplus c_{A2}) \dots (x_n \oplus c_{An})$ and $B = (x_1 \oplus c_{B1})(x_2 \oplus c_{B2}) \dots (x_n \oplus c_{Bn})$ be two POE terms have the fewest changes in polarity between EXOR-sums, which means $c_{Ai} = c_{Bi}$ for most of the case, for $1 \leq i \leq n$.

In this setting, A and B are referred to as the matching pair, and $C = (x_1 \oplus c_{C1})(x_2 \oplus c_{C2}) \dots (x_n \oplus c_{Cn})$ is referred to as the remaining POE term

2. Find which EXOR-sums have different coefficient inside the matching pair A and B (i.e. the change list). Let this change list be $M = x_j \dots x_k$, where $1 \leq j, k \leq n$
3. For the remaining POE term which is $C = (x_1 \oplus c_{C1})(x_2 \oplus c_{C2}) \dots (x_n \oplus c_{Cn})$, apply the same coefficient changes from the change list $M = x_j \dots x_k$ found in Step 2 to derive the completing POE term (i.e. XOR-ing with '1' all EXOR-sums in M that are in C). Let this completing POE term be (D) .

Once the completing term is found, the factorization can be done as follows:

- 1) Apply Proposition 1 to the matching pair A and B , we can find a single POE term (P_1) . Apply Proposition 1 to the remaining and completing POE terms (i.e. C and D), we can find in a single POE term (P_2) .
- 2) Apply Proposition 1 to P_1 and P_2 , we can find a single factored POE term (P_3) .

The proof is complete as the original function is now factorized into the following two POE terms: $P_3 \oplus D$.

As we can see, Proposition 2 factors three POE terms with N EXOR-sums into one POE term with $(N - 2)$ EXOR-sums and one POE terms with N EXOR-sums, which means Proposition 2 replaces the circuit with three $N \times N$ Toffoli gates with the circuit with one $(N - 2) \cdot (N - 2)$ Toffoli gate and one $N \times N$ Toffoli gate, so the quantum cost reduces from $2 \cdot (2^N - 3)$ to $(2^{N-2} - 3)$

3.2.3 The EPOEM-2 Algorithm:

The EPOEM-2 Algorithm converts a Boolean function of N input variables $(a, b, c, d, \dots)$ into EPOE form using a factorization method. The algorithm uses a strategy of

searching for pairs of minterms (POE terms) which have the common differing polarity sets in variables/terms. The set that has the most pairs (at least 2) is appended to a list called `Common_list`. The remainder set that holds all remainder minterms is again searched for pairs of minterms (POE terms) which have the common differing polarity set in variables/terms. The set that has the most pairs (at least 2) is again appended to the list `Common_list`. The same procedure is iteratively applied to the remainder set until no set can be found having more than two pairs of minterms (POE terms) that have the common differing polarity set in variables/terms. The last remainder set is now appended to the list called `Result_list`. Now apply Proposition 1 to each pair from the `Common_list` and append the results to the new `ON_set`. Apply the same procedure as above with the new `ON_set` until the `Common_list` becomes empty. Now for each set in the `Result_list`, if the set has more than three POE terms, the algorithm applies Proposition 2 to a **random** three POE terms from that set iteratively until the set holds less than three POE terms. The result from applying Proposition 2 is appended to the `Final_result`. If the result set has two POE terms, the algorithm applies Proposition 1 to that two POE terms and appends the result to the `Final_Result`. If the set has less than two POE terms, then the algorithm will append that set to the `Final_Result`. The same procedure is iteratively applied to each set in the `Result_list` and return the `Final_result`.

The algorithm for the three main procedures, discussed above, is shown below as well as the main function of the algorithm. An example of EPOEM-2 synthesis is presented subsequently.

A. Algorithm:

a) Find_Common_list (ON_set): This function returns the Common_list and

Best_Remainder set of the ON-set.

- Common_list is the list that holds all the sets of pairs of minterms (or POE terms) which have the common differing polarity set of variables/terms

For eg. ($a'b'cd$, $abcd$) and ($a'bc'd$, $ab'c'd$) are the two pairs belonging to the differing polarity set of $\{a, b\}$

- Best_Remainder is the set that hold all the minterms (or POE terms) which cannot form into pairs that have the common differing polarity set of variables/terms

-
- Create all pairs of minterms/POE_terms (s_i, s_j) , where s_i and s_j are two arbitrary minterms or POE_terms and $1 \leq i, j \leq n$ (n is the total number of minterms/POE_terms)
 - Find out the set of variables/POE_terms $(e_1, e_2, \dots, e_k)$ that is in differing polarity in each pair of minterms/POE_terms, where $1 \leq e_k \leq l$ (l is the length of minterm/POE_term)
 - Record all the **common** differing polarity set of variables/POE_terms $(e_1, e_2, \dots, e_k)$
 - Find the set that has the most pairs (at least 2 pairs), which have the common set of variables/POE_terms $(e_1, e_2, \dots, e_k)$ and append that set in the Common_list
 - Remainder_set = ON_set - M(Common_list)
 - Redo step i. to iv. with the Remainder_set until it cannot find any more set to put in the Common_list
Common_list, Remainder_set = Find_Common_list (Remainder_set)
 - Best_Remainder = Remainder_set
-

The time complexity for creating all pairs is $O(C_2^N)$ with N is the number of minterms/POE terms in the input set, for finding the common differing polarity set is $O(N)$ with N is the number of common differing polarity variable/term and for

recursively finding Common_list is $O(\log(N))$ with N is the number of terms in the Remainder_set. So the overall time complexity of this process is $O(C_2^N)$

- b) Find_Result_list(Common_list, Best_remainder): This recursive function will call a Find_Common_list() function to return the Result_list, the list that contains all the Best_Remainder sets after each Find_Common_list() call.

```

If Common_list = []
    Result_list.append(Best_remainder)
For a set in Common_list:
    For each pair in a set:
        adf = fact2 (pair)          //fact2(): Proposition 1
        New_Onset.append(adf)
    New_Common_list, New_Best_remainder = Find_Common_List(New_Onset)
    Find_result_list(New_Common_list, new_best_remainder)

```

The time complexity for Proposition 1 is $O(1)$, for recursively finding the new Common_list is $O(C_2^N \cdot \log(N))$ with N is the number of terms in the New_Onset and for recursively finding the Result_list is $O(C_2^N \cdot \log(N) \cdot \log(N))$ with N is the number of terms in the New_Common_list. So the overall time complexity of this process is $O(C_2^N \cdot \log(N))$.

- c) Find_Result(Result_set)

```

//Result_set: A set in Result_list
Result := []
If len(Result_set) > 2:
    Triset := Random pick 3 terms in a_set
    Fact3_result := fact3(Triset) //fact3(): Proposition 2
    Result := concat(Result, "⊕",
                     Fact3_result.factored_expression)
    remainder := Result_set EXOR Fact3_result.factoredPOE
    areresult = Find_Result(remainder)
    Result := concat(Result, "⊕", areresult)
If len(Result_set) = 2:
    Fact2_result := fact2(Result_set)
    Result := concat(Result, "⊕", Fact2_result)
Else:
    Result := concat(Result, "⊕", Result_set)

```

The time complexity of this process is $O(N \cdot \log(N))$ with N is the number of terms in the Result_set

d) Main function of EPOEM-2 algorithm:

```

Common_List, Best_Remainder = Find_Common_list(ON_set)
Result_list = Find_Result_list (Common_List, Best_Remainder)

For each result_set in Result_list:
    Final_Result = Find_result (result_set)

Compute the cost of the result in Final_result

Return the Final_result and the quantum cost of its expression

```

The overall time complexity of the EPOEM-2 algorithms is $O(C_2^N \cdot \log(N))$ with N is the number of terms in the ON_set

B. Example 7:

Here is an example of EPOEM-2 synthesis of function $F_7(a, b, c, d, e) =$

$\sum m(1,3,4,6,9,11,12,13,14,18,21,24,31)$

1. ON_set = ['00001', '00011', '00100', '00110', '01001', '01011', '01100', '01101',
'01110', '10010', '10101', '11000', '11111']
2. Find all combinations of 2 minterms in ON_set, for e.g. ('00001' , '00011'),
('00100' , '00110'), ('01001' , '01100'), ...
3. Record which variables are changing polarity in each pair, e.g.

('00001' , '00011') : Change in variable d
('00100' , '00110') : Change in variable d
('01001' , '01100') : Change in variables c and e ,

4. Record the list that has the most pairs of changing polarity in the same variables.
For this example, that list is changing in variable b , c and e , which have 6 pairs, and this list is appended to the `Common_list`.
5. So, the `Common_list` = [('00001' , '01100'), ('00011' , '01110'), ('00100' , '01001'), ('00110' , '01011'), ('10010' , '11111'), ('10101' , '11000')]
6. And a `Best_Remainder` is the left over minterm, which is: ['01101'] and it is appended to the `Result_List`. So, the `Result_list` = [['01101']]
7. Now apply the Proposition 1 to each pair in the `Common_list`, e.g.
 - For ('00001' , '01100'): $adf1 = \bar{a}(b \oplus e)\bar{d}(c \oplus e)$
 - For ('00011' , '01110'): $adf2 = \bar{a}(b \oplus e)d(c \oplus e)$
 - ...
8. Append $adf1, adf2, \dots, adf6$ (the result of applying Proposition 1 to 6 pairs in `Common_list`) to the new `ON_set`
9. `ON_set` = [$\bar{a}(b \oplus e)\bar{d}(c \oplus e)$, $\bar{a}(b \oplus e)d(c \oplus e)$,
 $\bar{a}(b \oplus e \oplus 1)\bar{d}(c \oplus e)$, $\bar{a}(b \oplus e \oplus 1)d(c \oplus e)$,
 $a(b \oplus e \oplus 1)d(c \oplus e \oplus 1)$, $a(b \oplus e)\bar{d}(c \oplus e \oplus 1)$]
10. Apply the same methods like in step 2 to 4 to get a new `Common_list` and a new `Best_Remainder`.
11. `Common_list`: [$(\bar{a}(b \oplus e)\bar{d}(c \oplus e) ; \bar{a}(b \oplus e \oplus 1)d(c \oplus e))$, $(\bar{a}d(c \oplus e) ; \bar{a}(b \oplus e \oplus 1)\bar{d}(c \oplus e))$, $(a(b \oplus e \oplus 1)d(c \oplus e \oplus 1) ; a(b \oplus e)\bar{d}(c \oplus e \oplus 1))$]. All the pairs have $(b \oplus e)$ and d in different polarity

12. Best_Remainder = [] (no POE terms left)
13. Apply Proposition 1 to the three pairs in Common_list:
 - For $(\bar{a}(b \oplus e)\bar{d}(c \oplus e); \bar{a}(b \oplus e \oplus 1)d(c \oplus e))$: $adf7 = \bar{a}(b \oplus d \oplus e)(c \oplus e)$
 - For $(\bar{a}d(c \oplus e); \bar{a}(b \oplus e \oplus 1)\bar{d}(c \oplus e))$: $adf8 = \bar{a}(b \oplus d \oplus e \oplus 1)(c \oplus e)$
 - For $(a(b \oplus e \oplus 1)d(c \oplus e \oplus 1); a(b \oplus e)\bar{d}(c \oplus e \oplus 1))$: $adf9 =$
 $a(b \oplus d \oplus e)(c \oplus e \oplus 1)$
14. Append $adf7, adf8, adf9$ to the new ON_set
15. $ON_set = [\bar{a}(b \oplus d \oplus e)(c \oplus e), \bar{a}(b \oplus d \oplus e \oplus 1)(c \oplus e),$
 $\bar{a}(b \oplus d \oplus e)(c \oplus e \oplus 1)]$
16. Because the new ON_set has only 3 terms the algorithm cannot find at least 2
 pairs that have different polarities in the same terms/variable. So there is no
 Common_list, and Best_Remainder = ON_set = $[\bar{a}(b \oplus d \oplus e)(c \oplus e),$
 $\bar{a}(b \oplus d \oplus e \oplus 1)(c \oplus e), a(b \oplus d \oplus e)(c \oplus e \oplus 1)]$
17. Append Best_Remainder to the Result_list
18. $Result_list = [['01101'], [\bar{a}(b \oplus d \oplus e)(c \oplus e),$
 $\bar{a}(b \oplus d \oplus e \oplus 1)(c \oplus e), a(b \oplus d \oplus e)(c \oplus e \oplus 1)]]$
19. Result_list has 2 sets: $['01101']$ and $[\bar{a}(b \oplus d \oplus e)(c \oplus e),$
 $\bar{a}(b \oplus d \oplus e \oplus 1)(c \oplus e), a(b \oplus d \oplus e)(c \oplus e \oplus 1)]$
20. Set $['01101']$ only has one item, so Result = $\bar{a}bc\bar{d}e$
21. Set $[\bar{a}(b \oplus d \oplus e)(c \oplus e), \bar{a}(b \oplus d \oplus e \oplus 1)(c \oplus e),$
 $\bar{a}(b \oplus d \oplus e)(c \oplus e \oplus 1)]$ has 3 elements, apply Proposition 2 to this set:

$$Fact3_{result} = (a \oplus c \oplus e) \oplus a(b \oplus d \oplus e \oplus 1)(c \oplus e \oplus 1)$$

22. So, $Final_result = \bar{a}bcde \oplus (a \oplus c \oplus e) \oplus a(b \oplus d \oplus e \oplus 1)(c \oplus e \oplus 1)$

which has the quantum cost of 95

23. The corresponding circuit is given in Figure 15

Figure 15. Circuit realization of function F_7 using EPOEM-2 algorithm.

Compare our EPOEM-2 result from Figure 15 with the circuit produced by Exorcism-4, $F_7 = a' \oplus b'e \oplus b'c' \oplus cde \oplus a'bc'e' \oplus ac'd'e' \oplus a'b'cde$ and its circuit realization that has a quantum cost of 143 (see Figure 16). It should be appreciated that EPOEM-2 result has a cost reduced by 33.5% with respect to the Exorcism-4 result. Exorcism-4 has a higher cost because of using multi-input Toffoli gates. Using gates with smaller numbers of inputs is the main reason of advantage of EPOE circuits over ESOP circuits (used by most authors) or various canonical types of Reed-Muller circuits, used by some authors.

Figure 16. Standard ESOP-like circuit realization of function $F_7 = a' \oplus b'e \oplus b'c' \oplus cde \oplus a'bc'e' \oplus ac'd'e' \oplus a'b'cde$ with only Toffoli and NOT gates

3.2.4 Experimental results:

EPOEM-2 program has been implemented in Python and tested extensively on Unix and Windows workstations. The experimental results below have been received on a 2.9 GHz Intel Core i7 PC under Microsoft Windows 8.1.

To verify and compare my above-presented algorithm, several single-output benchmark functions were taken from Revlib's page [44], Maslov's page [45] and [14] for synthesis testing. A comparison of the results from EPOEM-1s [14], EPOEM-2, EXORCISM-4 [9] is shown in Table 5. The EPOEM-2 methods can only synthesize up to 16 variables because the EPOEM-2 program creates too many combinations which consumes a lot of resources.

The algorithms presented so far were all heuristic and exhaustive, although the search spaces were much restricted in size thanks to my theorems, methods and heuristics. The fundamental question thus arises which is common to this type of algorithms – “how actually good are the heuristic algorithms, how far are the produced by them solutions with respect to the absolutely minimal, proven, so-called “exact minimum” solutions. To achieve this goal authors of SOP tools compare their algorithm to exact minimum solutions obtained from exact minimizers such as Quine-McCluskey based ones. It is well known that the area of EXOR-based synthesis is much more difficult than the area of SOP-based synthesis and for ESOP the exact algorithms have been found [91, 92] for not more than 6 inputs functions. The paper [91] created the first general method to solve this kind of problems, which was next improved by several authors but with not much numerical improvement in terms of the number of functions that can be exactly minimized. In order to evaluate how far my solutions are from the exact minimum solutions I have done an additional comparison of results from my EPOE tools with the results from the exact ESOP minimum

tool which is currently under development by PSU student Sanjay Sharma. His tool is a parallel program based on a standard Helliwell's Decision Function [91]. The method is based on even-odd covering based on decision function H and is not discussed in detail here, the reader can find all information in [91]. I created a new algorithm EPOE-EXACT using an extension of the standard Helliwell's function [91] to a function in which the selection functions g_i are created not only for 3^n product terms of ESOP but for all possible EPOE terms. My algorithm is thus a complete exhaustive tree search with no heuristics that is restricted only by the depth of search. In contrast to the original algorithm of Helliwell that minimizes the number of g_i functions, my version minimizes the quantum cost. Since the numbers of g_i functions for EPOE tool are too large compare with ESOP tool (51 vs. 27 for 3 variables function, 307 vs. 81 for 4 variables function, 2451 vs. 243 for 5 variables functions) which creates a very large search-space when select the combinations of g_i functions, the EPOE-EXACT tool can only synthesize for functions up to 4 variables while the ESOP tool can work up to 6 variables functions. And because of this reason, the EPOE-EXACT tool can only use to synthesize single output functions but not the multiple output function (a (3 input – 3 output) or (4 input – 4 output) functions which transform into a 6 input – 1 output functions or 8 input – 1 output functions which is over the limit). So this EPOE-EXACT tool is only used to find the minimum results of 4 variables single output benchmark functions and compare the results with the EPOEM-1s and EPOEM-2 algorithms to see how far are the produced by them solutions with respect to the exact minimum solutions. The results are shown in Table 5 (column (5)). The best quantum cost is shown in bold.

Observe please, that in my evaluation I took functions from all well-known and popular sources used by nearly all people in this area to be able to compare my results to the absolutely top results currently known in the world. I have done three types of evaluation:

- (1) Based on a high number of industrial and academic benchmark functions.
- (2) Based on statistical analysis based on randomization and testing my algorithm on some random benchmark functions as shown in Figure 17.
- (3) Based on comparison with exact minimum solutions of ESOP and EPOE obtained from an experimental tool and its modification.

Concluding, my method of evaluating results includes several independent methods and is more comprehensive than those used by all other authors that publish results in the sub-area of reversible and quantum logic synthesis.

Table 5. Benchmark functions synthesized with EXORCISM-4 (3) vs. EPOEM-1S (1) vs. EPOEM-2 (2) vs. EXACT MINIMUM ESOP (4) vs. EXACT MINIMUM EPOE EPOE-EXACT (5)

Function	Quantum Cost					Function	Quantum Cost				
	EPOEM-1s	EPOEM-2	(3)	(4)	(5)		EPOEM-1s	EPOEM-2	(3)	(4)	(5)
lt41	26	29	41	42	26	lt51	112	115	142		
lt42	49	65	50	50	49	lt52	92	109	104	106	
lt43	33	41	41	39	33	5alu_9	39	55	52	119	
lt44	29	37	58	62	29	5ex2_151	93	95	143		
lt45	49	49	64	67	49	5ex3_152	69	71	84	84	
4gt4_20	45	51	45	45	45	5majority	112	119	149	149	
4gt5_21	16	16	16	16	16	5rd53f1	69	73	177	185	
4gt10_22	36	36	36	41	36	5rd53f2	25	69	88		
4gt11_23	5	5	5	5	5	5xor5	9	9	9	13	
4gt12_24	38	38	44	38	38	7con1f1	119	139	141		
4gt13_25	13	13	13	13	13	7con2f2	60	65	68		
4mod5_8	13	13	28	28	13	7rd73f1	50	255	211		
4sf_232	27	38	38	40	27	7rd73f2	7	13	19		
lt61	187	207	293			7rd73f3	203	485	1337		
sym6	136	199	857			10sym		13567	50378		
8newill	684	875	1239			10sao2f1		4733	5090		
8newtag	483	555	683			10sao2f2		3183	8156		
8rd84f1	60	821	277			10sao2f3		7147	8285		
8rd84f2	8	15	24			10sao2f4		7265	10727		
8rd84f3	509	509	509			11cm152a		229	232		
8rd84f4	333	1995	4824			14co14		65763	229334		
9life		3671	12682			16ryy6		55435	171580		
9max46		4919	14082								
9sym9		2437	20968

a) Function 5ex3 with best quantum cost of 71

b) Function 6sym with best quantum cost of 199

c) Function 9sym with best quantum cost of 2437

Figure 17. Frequency distribution of quantum cost results from 50 random runs using EPOEM-2 to synthesize functions: a) 5ex3, b) 6sym, c) 9sym

3.3 Discussion about two methods:

As shown in Table 5, compared with EXORCISM-4, which is considered the best ESOP synthesis tool and has been used for over 10 years, and the exact minimum ESOP tool over many benchmark functions, EPOEM-1s consistently produced solutions of equal or lower quantum cost with improvements ranging up to typically 50%, and in some cases up to 85%. EPOEM-2 algorithm (using Boolean factorization), which cannot

produce the good solution as EPOEM-1s algorithm, but compared with EXORCISM-4, still was able to produce solutions of equal or lower quantum costs for 41 out of 46 benchmark functions. Compared with the exact minimum the EPOE tool for the function of 4 variables, EPOEM-1s consistently produced solutions of equal quantum cost for all benchmark functions, while EPOEM-2 produced solutions of equal quantum cost for 7/13 functions and higher quantum cost for 6/13 functions ranging up to 33%. The more detailed comparison of EPOEM-1 and EPOEM-2 algorithm is shown in Table 6.

Table 6. Comparison of EPOEM-1 and EPOEM-2 method

Method	Pros	Cons
EPOEM-1 (using template library)	<ul style="list-style-type: none"> • Simple algorithm, always give good results 	<ul style="list-style-type: none"> • Need to have a library calculated in advanced • Library takes longer time and bigger disk space to generate as the number of variables increase • The algorithm can only work up to 8 variables
EPOEM-2 (using Boolean factorization)	<ul style="list-style-type: none"> • The algorithm can work up to 16 variables • The result is calculated faster 	<ul style="list-style-type: none"> • The result is not optimize like EPOEM-1

Chapter 4: SYNTHESIS OF EPOE CIRCUITS FOR INCOMPLETELY SPECIFIED SINGLE OUTPUT BOOLEAN FUNCTIONS

The reversible logic synthesis of incompletely specified functions takes as input a function that contains don't cares and generates a reversible circuit. Formally, from the mathematical point of view it is not a function but a relation, so my approach can be called "minimization of Boolean Relations". However, because the name "Boolean function with don't cares" or "incompletely specified Boolean function" are commonly used in engineering literature, I will stick to this popular name. Observe, that in the area of reversible and quantum circuit synthesis absolute most of the current synthesis methods deal with only completely specified reversible logic functions. Up to now, "don't cares" cannot be handled efficiently by these methods. The problem of synthesizing incompletely specified reversible functions is of importance in many areas like 1) the design of incomplete oracles for machine learning applications 2) design of blocks included in larger oracles or spectral transforms where "don't cares" occur similar to classical logic network design and 3) reversible state machine design and quantum automata design. For instance "don't cares" are introduced when realizing the excitation functions of flip-flops such as JK or SR (all these concepts have been generalized to quantum automata with permutative excitation and output functions). Many real-life machines lead to excitation and output functions with very many don't cares because of the following properties: (a) number of states is other than 2^k states which causes functions for all other states encoded by don't cares, (b) there are many don't cares in original transition and output functions, (c) some very good encodings (such as one-hot codes) with non-minimal codes exist that have highly

non-minimal numbers of qubits. Similarly “don’t cares” are common in both synchronous-like and asynchronous-like realizations of reversible and quantum automata.

In this dissertation, I use the template matching method as introduced in Section 3.1 with some modifications to synthesize a single output incompletely specified functions (i.e. functions with don’t cares).

The EPOEM-1-DC algorithm converts a Boolean function of N input variables ($a, b, c, d, \dots$) into EPOE form using a template matching method. The templates are specific ordered sets of minterms that can be used in EPOE synthesis as in Table II (same as EPOEM-1). The algorithm uses a strategy of searching for a POE template (T_i) which satisfies two conditions:

- i) The template intersects the function’s ON-set and don’t care set (DC_set) in over $2/3$ of the template’s minterms ($M(T_i)$), more formally: $|M(T_i) \cap (ON \cup DC)| > \frac{2}{3} |M(T_i)|$,
- ii) The matching template is EXOR-ed with the function’s ON-set and DC_set to produce a remainder function with **a smaller ON-set**. This means that the selected template must be such that the number of ‘1s’ (minterms in the ON_set) covered by the template is greater than the number of ‘0s’ covered by that template).

This procedure is iteratively applied until the remainder function becomes the empty ON-set. The algorithm requires a library of POE templates calculated in advance like the EPOEM-1 algorithm. The algorithms for EPOEM-1-DC follow.

4.1 EPOEM-1-DC Algorithm:

The EPOEM-1-DC Algorithm which uses the single expression template library is shown below. An example of EPOEM-1s synthesis is presented subsequently.

```
k := 2/3
ON_set := list of ON-set minterms
DC_set := list of don't care minterms
n := number of input variables
level := 0 //level in the library of templates
result := NIL

Checkset := ON_set XOR DC_Set

// Check if full map match:
If Checkset.length() > k.power(2, n)
  New_Onset = set1 - Checkset
  If New_Onset.length() < ON_set.length()
    result := "1" //NOT gate
    ON_set := New_Onset
    Checkset := ON_set XOR DC_Set

level := level + 1

While ON_set.length() > 0
  If Checkset.length() < k.power(2, n - level)
    level := level + 1
  Else
    For templates in library at current level:
      Find all templates where the intersection of the template and
        the Checkset covers at least 2/3 of the template
    If such templates exist:
      New_Onset = (ON_set - Template) XOR (Template - Checkset)
      If New_Onset.length() < ON_set.length()
        Choose the template that has the smallest number of
          minterms in the New_Onset, in the case of tie
          select randomly among the best
        Result := Result XOR Template_name
        ON_set := New_Onset
        Checkset := ON_set XOR DC_Set
      Else
        level := level + 1

Return result
```

The time complexity for algorithm EPOEM-1-DC is the same as algorithm EPOEM-1s which is $O(N)$, with N is the number of templates in the library, because it searches the hash table library.

4.2 Examples of EPOEM-1-DC:

Below a complete EPOEM-1-DC synthesis will be performed on the function $F_8(A, B, C, D) = \sum m(0,1,4,11,15) + d(2,3,6,7,14)$. Assume that the four-variable POE full expression template library has been calculated in advance.

- The ON_set is initialized to the ON-set = {0000, 0001, 0100, 1011, 1111}.
- $DC_set = \{0010, 0011, 0110, 0111, 1110\}$
- The Checkset = $ON_set \oplus DC_set = \{0000, 0001, 0010, 0011, 0100, 0110, 0111, 1011, 1110, 1111\}$
- At $level = 0$ the *Checkset* has only 10 minterms, and since $10 < (2/3)2^4$ the function is not negated. Consequently a “1” is not appended to the *result*.
- At $level = 1$ a POE template library search is performed and the templates $(a \oplus I)$, (d) are found to intersect with 7 minterms in the *Checkset*, and the template $(a \oplus c \oplus I)$ is found to intersect with 6 minterms in the *Checkset* which is shown in Figure 18. Since 7 and 6 $> (2/3)2^3$ so all templates are acceptable.
- With templates $(a \oplus I)$, (d) and $(a \oplus c \oplus I)$ selected, $(ON_set - template)$ is EXORed with $(template - Checkset)$, and the new ON_set becomes {0101, 1011, 1111}, {0000, 0001, 0100, 1010} and {0101, 1010} respectively. So with the template $(a \oplus c \oplus I)$ selected, the smallest ON_set is created {0101, 1010}. Therefore, template $(a \oplus c \oplus I)$ is chosen and its expression is appended to the *result*.
- The new *Checkset* becomes {0010, 0011, 0110, 0111, 1110, 0101, 1010}.
- At $level = 2$ a POE template library search is performed and the only acceptable template found is $(c)(d \oplus 1)$, which intersects with 4 minterms in the *Checkset* and creates the

new smaller ON_set {0101} as shown in Figure 19. So the expression $(c)(d \oplus 1)$ is appended to the *result*.

- The new *Checkset* becomes {0010, 0011, 0110, 0111, 1110, 0101}.
- At *level* = 3 a POE template library search is performed and many acceptable templates are found with equal quantum cost and with 2 intersecting minterms, all of them create an empty ON_set . The template $(a \oplus 1)(b)(d)$ is **randomly** selected and is shown in Figure 20. Its expression is appended to the *result*.
- The new ON_set becomes { } which completes the synthesis. The final value of the *result* is as follows:

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (c)(d \oplus 1) \oplus (a \oplus 1)(b)(d)$$

Figure 18. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC.

Figure 19. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC.

Figure 20. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC.

Discussion: As we can see from Figure 19, instead of choosing a template from level 2 and jumping down to level 3, a POE template library search is performed and the template $(a \oplus b)(b \oplus c)(c \oplus d)$ is found, which intersects with 2 minterms in the Checkset and creates an empty ON_set . So the template $(a \oplus b)(b \oplus c)(c \oplus d)$ is selected, as shown in Figure 21. Its expression is appended to the result. The new final result is as follows:

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (a \oplus b)(b \oplus c)(c \oplus d)$$

Figure 21. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC.

So, the EPOE-1-DC can be extended to EPOE-1-DC-tree to get the minimum solution. EPOE-1-DC-tree will find all possible templates that:

- (i) intersect the function's ON-set and don't care set in over $\frac{1}{2}$ of the template's minterms
- (ii) EXOR-ed with the function's ON-set and DC_set the templates produce a remainder function with **a smaller ON-set**.

EPOE-1-DC-tree algorithm finds all such templates from the current level (start at level 0) to the highest level (level n-1) and its respective remainder function. This procedure is

iteratively applied until the remainder function becomes the empty ON-set. The algorithm compares the quantum costs of all the results and chooses one with the lowest cost. The algorithm requires a library of POE templates calculated in advance, like the EPOEM-1 algorithm. The algorithms for EPOEM-1-DC-tree follow.

4.3 EPOE-1-DC-tree algorithm:

```

ON_set := input(ON-set)           // input function to be synthesized
DC_set := input(DC-set)          // don't care set
n := number of input variables
Checkset := ON_set EXOR DC_set

// EPOEM1DCtree will aggregate all possible POE templates of a function
Check_set
// in a list named Final_Result
EPOEM1DC(Check_set, ON_set, level, root)
    m := number of minterms in ON_set
    if m < 1
        return root
    else if level < n
        for template in a level
            match = find intersect of checkset and mask
            if (length of match) > 2/3 * 2^(n-level)
                newonset = (onset - mask) xor (mask - checkset)
                if length of newonset < length of onset
                    save the match

            if match found
                find expression of the template
                if mask name is in root go to next level
                newcheckset = newonset union dcset
                add current satisfied expression to root

                findresult (newonset, newcheckset, level,
                                newroot)
            else go to next level

```

Main function of EPOEM-1-DC-tree algorithm:

```

Final_result := EPOEM1DC(Check_set, ON_set, 0, [])

Compute the cost of each result in Final_result
Select the expression with the lowest cost as the solution

```

The time complexity for algorithm EPOEM-1-DC-tree is $O(N \cdot \log(N))$, with N is the number of templates in the library, because it constructs a solution B-tree based on the hash table template library.

4.4 Examples of EPOEM-1-DC-tree application:

Below a complete EPOEM-1-DC-tree synthesis will be performed on the function $F_8(a, b, c, d) = \{0, 1, 4, 11, 15\} + d \{2, 3, 6, 7, 14\}$. Assume that a four-variable POE single expression template library has been calculated in advance.

- The ON_set is initialized to the ON-set = $\{0000, 0001, 0100, 1011, 1111\}$.
- $DC_set = \{0010, 0011, 0110, 0111, 1110\}$
- The Checkset = $ON_set \text{ EXORed } DC_set = \{0000, 0001, 0010, 0011, 0100, 0110, 0111, 1011, 1110, 1111\}$
- At $level = 0$ the *Checkset* has only 10 minterms, and since $10 < (2/3)2^4$ the function is not negated. Consequently a “1” is not appended to the *result*.
- Now a POE template library search is performed for all other level from $level = 1$ to 3
 - ❖ At $level = 1$ a POE template library search is performed and the templates $(a \oplus 1)$ and (d) are found to intersect with 7 minterms in the *Checkset*, and the templates $(a \oplus c \oplus 1)$ is found to intersect with 6 minterms in the *Checkset* which is shown in Figure 22. Since 7 and $6 > (2/3)2^3$ so all templates are acceptable.

With templates $(a \oplus 1)$, (d) and $(a \oplus c \oplus 1)$ is selected, $(ON_set - template)$ is EXORed with $(template - Checkset)$, the new ON_set becomes $\{0101, 1011, 1111\}$, $\{0000, 0001, 0100, 1010\}$ and $\{0101, 1010\}$ respectively which is all smaller than the $ON_set = \{0000, 0001, 0100, 1011, 1111\}$.

The new *Checkset* becomes {0010, 0011, 0110, 0111, 1110, 0101, 1010} with template $(a \oplus c \oplus I)$ being selected.

The new *Checkset* becomes {0010, 0011, 0110, 0111, 1110, 0101, 1010, 1011} with template $(a \oplus I)$ being selected.

The new *Checkset* becomes {0000, 0001, 0010, 0011, 0100, 0110, 0111, 1110, 1010} with template (d) being selected.

- With *Checkset* = {0010, 0011, 0110, 0111, 1110, 0101, 1010}

- At *level* = 1 a POE template library search is performed and there is no satisfied template which: (i) intersects the *Checkset* in over $\frac{1}{2}$ of the template's minterms, (ii) EXOR-ed with the *Checkset* to produce a remainder function with a **smaller ON-set**

- At *level* = 2 a POE template library search is performed and the acceptable template found is $(c)(d \oplus 1)$, which intersects with 4 minterms in the *Checkset* and creates the new smaller ON_set {0101}, as shown in Figure 23 (left branch).

Therefore the expression $(c)(d \oplus 1)$ is appended to the result.

- The new *Checkset* becomes {0010, 0011, 0110, 0111, 1110, 0101}.

- At *level* = 2 a POE template library search is performed and there is no satisfied template which (i) intersects the *Checkset* in over $\frac{1}{2}$ of the template's minterms and (ii) EXOR-ed with the *Checkset* produces a remainder function with a **smaller ON-set**.

- At *level* = 3 a POE template library search is performed and many acceptable templates are found with equal quantum costs and with 2 intersecting

minterms; all of them create also an empty ON_set . The template $(a \oplus 1)(b)(d)$ is randomly selected and is shown in Figure 23 (left branch). Its expression is appended to the *result*.

➤ The new ON_set becomes $\{ \}$ which completes the synthesis for this branch.

The final value of the *result* is as follows:

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (c)(d \oplus 1) \oplus (a \oplus 1)(b)(d)$$

➤ This *result* is stored in *Final_result_list*.

- At $level = 3$, a POE template library search is performed and the template $(a \oplus b)(b \oplus c)(c \oplus d)$ is found, which intersects with 2 minterms in the Checkset and creates the empty ON_set . So the template $(a \oplus b)(b \oplus c)(c \oplus d)$ is selected and is shown in Figure 23 (right branch). Its expression is appended to the result. The final value of the *result* is as follows:

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (a \oplus b)(b \oplus c)(c \oplus d)$$

This *result* is stored in *Final_result_list*.

- Apply the same routine as above with Checkset = {0010, 0011, 0110, 0111, 1110, 0101, 1010, 1011} and Checkset = {0000, 0001, 0010, 0011, 0100, 0110, 0111, 1110, 1010}, some others results are found such as:

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (a \oplus b)(b \oplus c)(c \oplus d) \quad \text{or}$$

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (a \oplus b)(b \oplus c)(c \oplus d)$$

All the *results* are stored in the *Final_result_list*.

Figure 22. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC-tree at $level = 1$

Figure 23. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC-tree at $level = 2$ and $level = 3$

❖ At $level = 2$ a POE template library search is performed and the templates $(a \oplus 1)(c \oplus d \oplus 1)$ and $(a \oplus b \oplus 1)(a \oplus c \oplus 1)$ are found to intersect with 4 minterms in the *Checkset*, and the template $(a \oplus 1)(c \oplus 1)$ is found to intersect with 3 minterms in the

Checkset, which is shown in Figure 24. Since both 4 and $3 > (2/3)2^2$ then all templates are acceptable.

With templates $(a \oplus 1)(c \oplus d \oplus 1)$, $(a \oplus b \oplus 1)(a \oplus c \oplus 1)$ and $(a \oplus 1)(c \oplus 1)$ selected, $(ON_set - template)$ is EXORed with $(template - Checkset)$, the new *ON_set* becomes $\{0001\}$, $\{0100, 1011\}$ and $\{0101, 1011, 1111\}$ respectively, which are all smaller than the $ON_set = \{0000, 0001, 0100, 1011, 1111\}$.

The new *Checkset* becomes $\{0010, 0011, 0110, 0111, 1110, 0001\}$ when template $(a \oplus 1)(c \oplus d \oplus 1)$ is selected.

The new *Checkset* becomes $\{0010, 0011, 0110, 0111, 1110, 0100, 1011\}$ when template $(a \oplus b \oplus 1)(a \oplus c \oplus 1)$ is selected.

The new *Checkset* becomes $\{0010, 0011, 0110, 0111, 1110, 0101, 1011, 1111\}$ when template $(a \oplus 1)(c \oplus 1)$ is selected.

- Apply the same routine as above with *Checkset* = $\{0010, 0011, 0110, 0111, 1110, 0001\}$, $\{0010, 0011, 0110, 0111, 1110, 0100, 1011\}$ and $\{0010, 0011, 0110, 0111, 1110, 0101, 1011, 1111\}$, some others results are found as shown in Figure 25 as follows:

$$F_8(a, b, c, d) = (a \oplus 1)(c \oplus d \oplus 1) \oplus (a \oplus 1)(b \oplus 1)(d) \text{ or}$$

$$F_8(a, b, c, d) = (a \oplus b \oplus 1)(a \oplus c \oplus 1) \oplus (a \oplus b)(b \oplus c)(b \oplus d) \text{ or}$$

$$F_8(a, b, c, d) = (a \oplus 1)(c \oplus 1) \oplus (a)(c)(d) \oplus (a \oplus 1)(b)(d)$$

All the *results* are stored in the *Final_result_list*.

Figure 24. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC-tree at *level = 2*

Figure 25. Karnaugh maps that illustrate the Checkset function evolution in EPOEM-1-DC-tree

❖ Same as *level = 2*, apply the same routine for *level = 3*, some others results are found is shown in Figure 26 as follows:

$$F_8(a, b, c, d) = (a \oplus 1)(b \oplus 1)(c \oplus 1) \oplus (a)(c)(d) \oplus (a \oplus 1)(b)(d \oplus 1) \text{ or}$$

$$F_8(a, b, c, d) = (a)(c)(d) \oplus (a \oplus 1)(c \oplus 1)(d \oplus 1) \oplus (a \oplus 1)(b \oplus 1)(d) \text{ or}$$

$$F_8(a, b, c, d) = (a \oplus 1)(b \oplus 1)(d \oplus 1) \oplus (a)(c)(d) \oplus (a \oplus 1)(c \oplus 1)(b \oplus d)$$

All the *results* are stored in the *Final_result_list*.

Figure 26. Karnaugh maps that illustrate the Checkset function comparisons of different templates in EPOEM-1-DC-tree at *level = 3*

- Compare all the costs and select the EPOE expression with the minimum quantum cost.

The lowest cost is 21 which leads to the selection of the following EPOE expression:

$$F_8(a, b, c, d) = (a \oplus c \oplus 1) \oplus (a \oplus b)(b \oplus c)(c \oplus d)$$

4.5 Experimental Results:

EPOEM-1-DC programs have been implemented in Python and tested extensively on Unix and Windows workstations. The experimental results below have been received on a 2.9 GHz Intel Core i7 PC under Microsoft Windows 8.1.

To verify and compare EPOEM-1-DC and EPOEM-1-DC-tree algorithms, several incompletely specified single-output benchmark functions were created in Table 7 for synthesis testing. A comparison of the results from EPOEM-1-DC, EPOEM-1-DC-tree algorithms is shown in Table 8. The best quantum cost is shown in bold. On the other hand, I have also done statistical analysis based on randomization and testing my algorithm on some random benchmark functions as shown in Figure 27.

Table 7. Incompletely specified single output benchmark functions

	ON_set	DC_Set
DC41	{0, 1, 4}	{2,3,6,7}
DC42	{0, 1, 4, 11, 15}	{2, 3, 6, 7,14}
DC43	{0, 1}	{2, 3, 6, 7}
DC44	{0, 1, 4}	{2, 3, 6, 7, 13, 15}
DC45	{0, 1, 4, 5, 11, 12}	{2, 3, 6, 7, 8, 9}
DC46	{4, 6, 9, 10}	{5, 7, 11}
DC47	{0, 1, 4, 9, 15}	{2, 3, 6, 7}
DC48	{0, 1, 4, 5, 11, 12}	{2, 3, 6}
DC51	{0, 1, 2, 6, 12, 19, 31}	{2, 4, 7, 13, 15, 18, 20, 23, 25, 27}
DC52	{0, 1, 2, 6, 9, 12, 19,21, 26, 31}	{3, 4, 5, 7, 13, 15, 18, 20, 23, 25, 27}
DC61	{000---,010--0, 1001--, 1111--, --001-, 001111, 001101}	{01101-, 0010--, 0110--, 0111--}

Table 8. Benchmark functions synthesized with EPOEM-1-DC vs. EPOEM-1-DC-tree

Function	Results		Quantum Cost
dc41	EPOEM-1-DC-tree	$(a \oplus 1) \oplus (a \oplus 1)bd$	17
	EPOEM-1-DC	$(a \oplus 1) \oplus (a \oplus 1)(b \oplus 1)(c \oplus 1)$	19
dc42	EPOEM-1-DC-tree	$(a \oplus c \oplus 1) \oplus (a \oplus d)(c \oplus d)(b \oplus c)$	24
	EPOEM-1-DC	$(a \oplus c \oplus 1) \oplus (a \oplus d)(c \oplus d)(b \oplus c)$	24
dc43	EPOEM-1-DC-tree	$(a \oplus 1)(b \oplus 1)$	9
	EPOEM-1-DC	$(a \oplus 1)(b \oplus 1)$	9
dc44	EPOEM-1-DC-tree	$(a \oplus 1) \oplus bd$	8
	EPOEM-1-DC	$(a \oplus 1) \oplus (a \oplus 1)(b \oplus 1)$	8
dc45	EPOEM-1-DC-tree	$c \oplus a(b \oplus c)d \oplus 1$	17
	EPOEM-1-DC	$(a \oplus c \oplus 1) \oplus (a \oplus 1)(b \oplus 1)(c \oplus 1) \oplus 1$	24
dc46	EPOEM-1-DC-tree	$(a \oplus b) \oplus (c \oplus 1)(a \oplus d)(a \oplus b)$	22
	EPOEM-1-DC	$(a \oplus 1) \oplus (a \oplus 1)(b \oplus 1)(c \oplus 1)$	22
dc47	EPOEM-1-DC-tree	$(a \oplus 1) \oplus (a \oplus b \oplus c)d$	10
	EPOEM-1-DC	$(a \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)$	12
dc48	EPOEM-1-DC-tree	$(a \oplus 1) \oplus cd \oplus ab(c \oplus d \oplus 1)$	25
	EPOEM-1-DC	$(a \oplus 1) \oplus (a \oplus 1)(b \oplus 1) \oplus (a \oplus 1)(b \oplus 1)(c \oplus 1)$	37
dc51	EPOEM-1-DC-tree	$(b) \oplus (c \oplus d)(a)$	8
	EPOEM-1-DC	$(a \oplus b \oplus c \oplus d \oplus 1) \oplus (a \oplus 1)(c \oplus d \oplus 1)$	18
dc52	EPOEM-1-DC-tree (25m)	$(a \oplus 1) \oplus (d) \oplus (c)(d) \oplus (a \oplus e \oplus 1)(b \oplus c \oplus d) \oplus (b \oplus 1)(d)(e)$	37
	EPOEM-1-DC	$(a \oplus d \oplus 1) \oplus (a \oplus d \oplus e \oplus 1)(b \oplus c \oplus d) \oplus (b \oplus 1)(c) \oplus (b \oplus 1)(c \oplus 1)(c \oplus d)$	46
dc61	EPOEM-1-DC-tree (4 hours+)		110
	EPOEM-1-DC	$(b \oplus c) \oplus (a \oplus 1)(a \oplus b \oplus c) \oplus (d \oplus 1)(c \oplus e)(a \oplus c) \oplus (a \oplus 1)(c \oplus d \oplus f)(b \oplus c) \oplus (a \oplus 1)(d \oplus 1)(c \oplus f)(c \oplus e)(b \oplus c) \oplus 1$	123

As shown in Table 8, EPOEM-1-DC-tree algorithm always gives the best solution but takes much longer time to generate the result when the function has more than 5 variables compared to the EPOEM-1-DC algorithm and the exact minimum EPOE tool. Also compare to the exact minimum EPOE, EPOEM-1-DC produced solutions of equal or lower quantum cost for 7/8 benchmark functions and because the EPOEM-1-DC is built based on the EPOEM-1s method, so it gives the same result for all the completely specified benchmark functions, when compared to EPOEM-1s as shown in Table 9.

Table 9. Benchmark functions synthesized with EPOEM-1-DC vs. EPOEM-1S vs. EXORCISM-4

Function	Quantum Cost			Function	Quantum Cost		
	EPOE-1-DC	EPOEM-1s	EXORC-4		EPOEM-1-DC	EPOEM-1s	EXORC-4
lt41	26	26	41	lt51	112	112	142
lt42	49	49	50	lt52	92	92	104
lt43	33	33	41	5alu_9	39	39	52
lt44	29	29	58	5ex2_151	93	93	143
lt45	49	49	64	5ex3_152	81	81	84
4gt4_20	45	45	45	5majority	116	116	149
4gt5_21	16	16	16	5rd53f1	91	91	177
4gt10_22	36	36	36	5rd53f2	29	29	88
4gt11_23	5	5	5	5xor5	9	9	9
4gt12_24	38	38	44	lt61	183	183	293
4gt13_25	13	13	13	sym6_63	159	159	857
4mod5_8	13	13	28	7con1f1	119	119	141
4sf_232	27	27	38	7con2f2	60	60	68
8newill	680	680	1239	8rdf84f1	60	60	277
8newtag	483	483	683	8rd84f2	15	15	24
				8rd84f3	509	509	509
				8rd84f4	333	333	4824

a) Function dc42 with best quantum cost of 31

b) Function dc51 with best quantum cost of 18

c) Function dc52 with best quantum cost of 46

d) Function dc61 with best quantum cost of 126

Figure 27. Frequency distribution of quantum cost results from 50 random runs using EPOEM-1-DC to synthesize functions: a) dc42, b) dc51, c) dc52 and d) dc61

Chapter 5: SYNTHESIS OF EPOE CIRCUITS FOR MULTIPLE OUTPUT BOOLEAN FUNCTIONS

Not much research besides [93, 94, 95] was published about multi-output EXOR synthesis, the problem that is much more complex than the single output function minimization. To achieve the objective of how to synthesize the multiple output function for quantum and reversible circuits with EPOE type circuits, I am proposing two methods:

- The first method adapts the general concept of the Muller's method from [78] to EPOE circuits and additionally combines it with the shared cube approach method from [11].
- The second method uses the general EXOR lattice concept from [96] but applies it to EPOE circuits.

5.1 EPOEM-MO-1:

EPOEM-MO-1 algorithm, the algorithm created to synthesize multiple output functions, consists of the following three phases:

- Phase 1: Transforms the multiple output Boolean function to a single output Boolean function.
- Phase 2: Uses EPOEM-1-DC to find the EPOE expression for a function which is converted from Phase 1 and generates the EPOE expressions for each of the outputs from that expression.

- Phase 3: Simplifies each EPOE expressions and from that generates the POEs sub-lists to find the shared POE terms and then transforms the sub-lists to gate-lists.

The time complexity for phase 1 is $O(1)$, for phase 2 is $O(N)$ with N is the number of templates in the library because it uses EPOEM-1-DC algorithm and for phase 3 is $O(N)$ with N is the number of POE terms in the EPOE outputs' expressions because it process the whole list of POE terms to generate the sub-lists. So, the overall time complexity for EPOEM-MO-1 algorithm is $O(N)$ with N is the number of templates in the library.

5.1.1 Phase 1:

A. Algorithm:

Using Muller's method to transform the multiple output Boolean function to a single output Boolean function.

If we have a multiple output Boolean function $\{f_1(x_1, x_2, \dots, x_m), f_2(x_1, x_2, \dots, x_m), \dots, f_n(x_1, x_2, \dots, x_m)\}$, we transform it to a single output Boolean function $f(x_1, x_2, \dots, x_m, f_1, f_2, \dots, f_n)$ such that:

$$f(x_1, x_2, \dots, x_m, 1, 0, 0, \dots, 0) = f_1(x_1, x_2, \dots, x_m)$$

$$f(x_1, x_2, \dots, x_m, 0, 1, 0, \dots, 0) = f_2(x_1, x_2, \dots, x_m)$$

...

$$f(x_1, x_2, \dots, x_m, 0, 0, 0, \dots, 1) = f_n(x_1, x_2, \dots, x_m)$$

All the remaining outputs of $f = -$ (assigned don't cares)

B. Examples:

Example 1:

As shown in Figure 28 below, a three output function $\{f_1(a, b, c), f_2(a, b, c), f_3(a, b, c)\}$ is transformed to a single output function $f(a, b, c, f_1, f_2, f_3)$ such that:

- When $f_1 = 1$ and $f_2 = f_3 = 0$: $f(a, b, c, 1, 0, 0) = f_1(a, b, c)$ as shown in red color
 - When $f_2 = 1$ and $f_1 = f_3 = 0$: $f(a, b, c, 0, 1, 0) = f_2(a, b, c)$ as shown in blue color
 - When $f_3 = 1$ and $f_1 = f_2 = 0$: $f(a, b, c, 0, 0, 1) = f_3(a, b, c)$ as shown in green color
 - When others:
 - $(f_1 = f_2 = 1 \text{ and } f_3 = 0)$ or $(f_1 = f_3 = 1 \text{ and } f_2 = 0)$ or $(f_2 = f_3 = 1 \text{ and } f_1 = 0)$
 - $f_1 = f_2 = f_3 = 1$
 - $f_1 = f_2 = f_3 = 1$
- $f(a, b, c) = -$ (don't care) as shown in purple color

a b c	f ₁ f ₂ f ₃
0 0 0	1 1 1
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	0 0 0

a b c f ₁ f ₂ f ₃	f
0 0 0 1 0 0	1
0 0 1 1 0 0	0
0 1 0 1 0 0	0
0 1 1 1 0 0	0
1 0 0 1 0 0	1
1 0 1 1 0 0	1
1 1 0 1 0 0	1
1 1 1 1 0 0	1
0 0 0 1 0 0	1
0 0 1 1 0 0	0
0 1 0 1 0 0	1
0 1 1 1 0 0	1
1 0 0 1 0 0	0
1 0 1 1 0 0	0
1 1 0 1 0 0	1
1 1 1 1 0 0	1
0 0 0 1 0 0	1
0 0 1 1 0 0	1
0 1 0 1 0 0	0
0 1 1 1 0 0	1
1 0 0 1 0 0	0
1 0 1 1 0 0	1
1 1 0 1 0 0	0
1 1 1 1 0 0	1
---0 0 0	—
---1 1 0	—
---1 0 1	—
---0 1 1	—
---1 1 1	—

Figure 28. Truth table illustrates the transformation from Example 1.

Example 2:

As shown in Figure 29 below, a four output function $\{f_1(a, b, c, d), f_2(a, b, c, d), f_3(a, b, c, d), f_4(a, b, c, d)\}$ is transformed to a single output function $f(a, b, c, d, f_1, f_2, f_3, f_4)$ such that:

- When $f_1 = 1$ and $f_2 = f_3 = f_4 = 0$: $f(a, b, c, d, 1, 0, 0, 0) = f_1(a, b, c, d)$ as shown in red color
- When $f_2 = 1$ and $f_1 = f_3 = f_4 = 0$: $f(a, b, c, d, 0, 1, 0, 0) = f_2(a, b, c, d)$ as shown in green color
- When $f_3 = 1$ and $f_1 = f_2 = f_4 = 0$: $f(a, b, c, d, 0, 0, 1, 0) = f_3(a, b, c, d)$ as shown in blue color

- When $f_4 = 1$ and $f_1 = f_2 = f_3 = 0$: $f(a, b, c, d, 0, 0, 0, 1) = f_4(a, b, c, d)$ as shown in purple color

- When others:

$f(a, b, c, d) = -$ (don't care) as shown in brown color

a b c d	f_1	f_2	f_3	f_4
0 0 0 0	0	0	0	0
0 0 0 1	0	0	1	1
0 0 1 0	0	0	1	0
0 0 1 1	1	0	0	1
0 1 0 0	0	1	0	1
0 1 0 1	0	1	1	1
0 1 1 0	1	0	1	0
0 1 1 1	0	0	0	1
1 0 0 0	1	0	0	0
1 0 0 1	1	0	1	1
1 0 1 0	1	1	1	1
1 0 1 1	0	1	0	0
1 1 0 0	1	1	0	1
1 1 0 1	1	1	0	0
1 1 1 0	0	1	1	0
1 1 1 1	1	1	1	0

a b c d f_1 f_2 f_3 f_4	f
0 0 0 0 1 0 0 0	0
0 0 0 1 1 0 0 0	0
0 0 1 0 1 0 0 0	0
0 0 1 1 1 0 0 0	1
0 1 0 0 1 0 0 0	0
0 1 0 1 1 0 0 0	0
0 1 1 0 1 0 0 0	1
0 1 1 1 1 0 0 0	0
1 0 0 0 1 0 0 0	1
1 0 0 1 1 0 0 0	1
1 0 1 0 1 0 0 0	1
1 0 1 1 1 0 0 0	0
1 1 0 0 1 0 0 0	1
1 1 0 1 1 0 0 0	1
1 1 1 0 1 0 0 0	1
1 1 1 1 1 0 0 0	0
1 1 1 1 1 0 0 0	1

a b c d f_1 f_2 f_3 f_4	f
0 0 0 0 1 0 0	0
0 0 0 1 0 1 0 0	0
0 0 1 0 0 1 0 0	0
0 0 1 1 0 1 0 0	0
0 1 0 0 1 0 0	1
0 1 0 1 0 1 0 0	1
0 1 1 0 0 1 0 0	1
0 1 1 1 0 1 0 0	0
1 0 0 0 1 0 0	0
1 0 0 1 0 1 0 0	0
1 0 1 0 0 1 0 0	1
1 0 1 1 0 1 0 0	1
1 1 0 0 0 1 0 0	1
1 1 0 1 0 1 0 0	1
1 1 1 0 0 1 0 0	1
1 1 1 1 0 1 0 0	1

a b c d f_1 f_2 f_3 f_4	f
0 0 0 0 0 0 1 0	0
0 0 0 1 0 0 1 0	1
0 0 1 0 0 0 1 0	1
0 0 1 1 0 0 1 0	1
0 1 0 0 0 0 1 0	0
0 1 0 0 0 1 0	0
0 1 0 1 0 0 1 0	1
0 1 1 0 0 0 1 0	1
0 1 1 1 0 0 1 0	0
1 0 0 0 0 0 1 0	0
1 0 0 1 0 0 1 0	1
1 0 1 0 0 0 1 0	1
1 0 1 1 0 0 1 0	0
1 1 0 0 0 0 1 0	0
1 1 0 1 0 0 1 0	0
1 1 1 0 0 0 1 0	1
1 1 1 1 0 0 1 0	1

a b c d f_1 f_2 f_3 f_4	f
0 0 0 0 0 0 1	0
0 0 0 1 0 0 0 1	1
0 0 1 0 0 0 0 1	0
0 0 1 1 0 0 0 1	1
0 1 0 0 0 0 0 1	1
0 1 0 0 0 0 1	1
0 1 0 1 0 0 0 1	0
0 1 1 0 0 0 0 1	1
0 1 1 1 0 0 0 1	1
1 0 0 0 0 0 0 1	0
1 0 0 1 0 0 0 1	1
1 0 1 0 0 0 0 1	1
1 0 1 1 0 0 0 1	0
1 1 0 0 0 0 0 1	0
1 1 0 1 0 0 0 1	1
1 1 1 0 0 0 0 1	0
1 1 1 1 0 0 0 1	0

a b c d f_1 f_2 f_3 f_4	f
-----0 0 0 0	-
-----0 0 1 1	-
-----0 1 0 1	-
-----1 0 0 1	-
-----0 1 1 0	-
-----1 0 1 0	-
-----1 1 0 0	-
-----0 1 1 1	-
-----1 0 1 1	-
-----1 1 0 1	-
-----1 1 1 0	-
-----1 1 1 1	-

Figure 29. Truth table illustrates the transformation from Example 2.

5.1.2 Phase 2:

A. Algorithm:

First use EPOEM-1-DC method to find the EPOE expression for a single output Boolean function that was obtained in Phase 1.

Second, expand all the EPOE expressions of each output in the original multiple output function from that EPOE expression.

After using EPOEM-1DC to synthesis the stage_1's function, we got the EPOE expression like this:

$$f(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = c_0 \oplus c_i \cdot g_i \quad c_i = 0 \text{ or } 1, (i \geq 0)$$

With $g_i(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = h_1(x_1, x_2, \dots, x_n) \cdot h_2(x_1, x_2, \dots, x_n) \dots h_k(x_1, x_2, \dots, x_n)$,

where k is the arbitrary number

With $h(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) =$

$$c_0 \oplus (c_{n1}x_1 \oplus c_{n2}x_2 \oplus \dots \oplus c_{nn}x_n) \oplus (c_{m1}y_1 \oplus c_{m2}y_2 \oplus \dots \oplus c_{mm}y_m), \quad c_i = 0 \text{ or } 1, (0 \leq i \leq \max(m, n))$$

$$\text{Let } P = (c_{n1}x_1 \oplus c_{n2}x_2 \oplus \dots \oplus c_{nn}x_n)$$

$$\text{Let } Q = (c_{m1}y_1 \oplus c_{m2}y_2 \oplus \dots \oplus c_{mm}y_m)$$

$$h(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = c_0 \oplus P(x_1, x_2, \dots, x_n) \oplus Q(y_1, y_2, \dots, y_m), \quad c_i = 0 \text{ or } 1, (0 \leq i \leq \max(m, n))$$

- If $Q = 0$, then $h(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = c_0 \oplus P(x_1, x_2, \dots, x_n)$

$$\Rightarrow h(x_1, x_2, \dots, x_n) \in \{y_1, y_2, \dots, y_m\}$$

- If $Q = (y_i \oplus \dots \oplus y_j)$, then $h(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) =$

$$c_0 \oplus P(x_1, x_2, \dots, x_n) \oplus (y_i \oplus \dots \oplus y_j) \quad (\text{there exists at least one } c_m = 1)$$

$$\Rightarrow \begin{cases} c_0 \oplus P(x_1, x_2, \dots, x_n) \oplus 1 \in \{y_i, \dots, y_j\} \\ c_0 \oplus P(x_1, x_2, \dots, x_n) \in \text{Remainder of } y \text{ set } \{y_i, \dots, y_j\} \end{cases}$$

$$\text{For example: } h(x_1, x_2, \dots, x_5, y_1, y_2, y_3) = c_0 \oplus x_1 \oplus x_2 \oplus (y_1 \oplus y_3)$$

$$\Rightarrow \begin{cases} (c_0 \oplus x_1 \oplus x_2) \oplus 1 \in \{y_1, y_3\} \\ (c_0 \oplus x_1 \oplus x_2) \in y_2 \end{cases}$$

Then we will create g from the product of h_i , but we need to note two cases:

- If $P = 0$, then $h(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = c_0 \oplus Q(y_1, y_2, \dots, y_n)$

If we call this $h(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = c_0 \oplus Q(y_1, y_2, \dots, y_m) = T(y_1, \dots, y_m)$

Then

$$\Rightarrow g(x_1, x_2, \dots, x_n) = h_m \dots h_n \cdot T$$

- If $c_0 = 0 \Rightarrow (h_m \dots h_n) \in \text{set of } y \text{ in } T$
- If $c_0 = 1 \Rightarrow (h_m \dots h_n) \in \text{remainder set of } y \text{ in } T$

For example: $g(x_1, x_2, \dots, x_5, y_1, y_2, y_3) = (x_1 \oplus x_2)(x_3 \oplus x_5 \oplus 1)(y_1 \oplus y_3)$

$$\Rightarrow (x_1 \oplus x_2)(x_3 \oplus x_5 \oplus 1) \in \{y_1, y_3\}$$

$$g(x_1, x_2, \dots, x_5, y_1, y_2, y_3) = (x_1 \oplus x_2)(x_3 \oplus x_5 \oplus 1)(y_1 \oplus y_3 \oplus 1)$$

$$\Rightarrow (x_1 \oplus x_2)(x_3 \oplus x_5 \oplus 1) \in \{y_2\}$$

B. Examples:

❖ Let's apply the Phase 2 algorithm to the example 1 on Phase 1.

- Using EPOEM-1-DC to find the EPOE expression for f:

$$f = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b \oplus f_1 \oplus f_2 \oplus 1)(a \oplus c \oplus f_1 \oplus f_3 \oplus 1)$$

- Expand all the EPOE expressions of each output in the original multiple output function from that EPOE expression

$$f = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b \oplus f_1 \oplus f_2 \oplus 1)(a \oplus c \oplus f_1 \oplus f_3 \oplus 1) \quad \longrightarrow \quad \begin{cases} (a \oplus b \oplus c \oplus 1) \in f_1, f_2, f_3 \\ \begin{cases} (a \oplus b)(a \oplus c) \in f_1 \\ (a \oplus b)(a \oplus c \oplus 1) \in f_2 \\ (a \oplus b \oplus 1)(a \oplus c) \in f_3 \end{cases} \end{cases}$$

So,

$$f_1 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b)(a \oplus c)$$

$$f_2 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b)(a \oplus c \oplus 1)$$

$$f_3 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(a \oplus c)$$

❖ Let's apply the Phase 2 algorithm to the example 2 on Phase 1.

- Using EPOEM-1-DC to find the EPOE expression for f:

$$\begin{aligned} f = & (a \oplus c \oplus f_1 \oplus 1) \oplus (a \oplus d \oplus 1)(f_1 \oplus f_2 \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus f_1 \oplus f_4 \oplus 1) \oplus \\ & (a \oplus c \oplus d \oplus 1)(c \oplus f_1 \oplus 1)(f_2 \oplus f_3 \oplus f_4 \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)(f_1 \oplus f_2 \oplus f_4 \oplus 1) \oplus \\ & (a \oplus d \oplus 1)(b \oplus f_3 \oplus f_4 \oplus 1)(f_1 \oplus f_2 \oplus 1) \oplus (a \oplus 1)(d \oplus 1)(b \oplus f_3 \oplus 1)(f_1 \oplus f_2 \oplus f_4 \oplus 1) \oplus \\ & (a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b \oplus f_4 \oplus 1)(f_1 \oplus f_2 \oplus f_3 \oplus 1) \end{aligned}$$

- Expand all the EPOE expressions of each output in the original multiple output function from that EPOE expression.

$$(a \oplus c \oplus f_1 \oplus 1) \longrightarrow \begin{cases} (a \oplus c) \in f_1 \\ (a \oplus c \oplus 1) \in f_2, f_3, f_4 \end{cases}$$

$$(a \oplus d \oplus 1)(f_1 \oplus f_2 \oplus 1) \longrightarrow (a \oplus d \oplus 1) \in f_3, f_4$$

$$(a \oplus b \oplus 1)(c \oplus f_1 \oplus f_4 \oplus 1) \longrightarrow \begin{cases} (a \oplus b \oplus 1)(c \oplus 1) \in f_2, f_3 \\ (a \oplus b \oplus 1)(c) \in f_1, f_4 \end{cases}$$

$$(a \oplus c \oplus d \oplus 1)(c \oplus f_1 \oplus 1)(f_2 \oplus f_3 \oplus f_4 \oplus 1) \longrightarrow (a \oplus c \oplus d \oplus 1)(c) \in f_1$$

$$(a \oplus b \oplus 1)(c \oplus 1)(f_1 \oplus f_2 \oplus f_4 \oplus 1) \longrightarrow (a \oplus b \oplus 1)(c \oplus 1) \in f_3$$

$$(a \oplus d \oplus 1)(b \oplus f_3 \oplus f_4 \oplus 1)(f_1 \oplus f_2 \oplus 1) \longrightarrow (a \oplus d \oplus 1)(b) \in f_3, f_4$$

$$(a \oplus 1)(d \oplus 1)(b \oplus f_3 \oplus 1)(f_1 \oplus f_2 \oplus f_4 \oplus 1) \longrightarrow (a \oplus 1)(d \oplus 1)(b) \in f_3$$

$$(a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b \oplus f_4 \oplus 1)(f_1 \oplus f_2 \oplus f_3 \oplus 1) \longrightarrow (a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b) \in f_4$$

So,

$$f_1 = (a \oplus c) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus c \oplus d \oplus 1)(c)$$

$$f_2 = (a \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)$$

$$f_3 = (a \oplus c \oplus 1) \oplus (a \oplus d \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1) \oplus (a \oplus d \oplus 1)(b) \oplus (a \oplus 1)(d \oplus 1)(b)$$

$$f_4 = (a \oplus c \oplus 1) \oplus (a \oplus d \oplus 1) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus d \oplus 1)(b) \oplus (a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$$

5.1.3 Phase 3:

A. Algorithm:

First, use the following Boolean algebra identities to simplify the EPOE expressions obtained from Phase 2 of the algorithm:

- $A \oplus A = 0$
- $A \oplus \bar{A} = 1$
- $A \oplus AB = A\bar{B}$
- $A \oplus A\bar{B} = AB$

Second, apply shared cube approach method [11] to find the shared POE terms and then generate the gate-lists.

B. Examples:

❖ Using the Phase 2's example 1 expressions, we have:

$$f_1 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b)(a \oplus c)$$

$$f_2 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b)(a \oplus c \oplus 1)$$

$$f_3 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(a \oplus c)$$

- These expressions are already in the simplest form, cannot simplified anymore.
- The initial POE terms list is as follow:

POE terms:	f_1 f_2 f_3
$(a \oplus b \oplus c \oplus 1)$	1 1 1
$(a \oplus b)(a \oplus c)$	1 1 1

The POE terms $(a \oplus b)(a \oplus c)$, $(a \oplus b)(a \oplus c \oplus 1)$ and $(a \oplus b \oplus 1)(a \oplus c)$ are considered as one POE term in the list because they are in the same support family of POE $(a \oplus b)(a \oplus c)$.

Both POE terms $(a \oplus b)(a \oplus c)$ and $(a \oplus b \oplus c \oplus 1)$ are shared by all three functions f_1, f_2, f_3 . So, they are moved to sub-list 1:

POE terms:	f_1 f_2 f_3
$(a \oplus b \oplus c \oplus 1)$	1 1 1
$(a \oplus b)(a \oplus c)$	1 1 1

Sub-list 1

There is no POE terms left in the initial list, which completes the phase.

Let's check out the two circuit realizations below, Figure 30 is realized based on the expressions obtain from Phase 2 with the cost of 32 and Figure 31 is realized using Phase 3 algorithm with the cost of 22.

Figure 30. Circuit realization of Example 1 produced by EPOEM-MO-1 algorithm without shared-POE-term phase

Figure 31. Circuit realization of Example 1 produced by EPOEM-MO-1 algorithm with shared-POE-term phase

$$f_1 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b)(a \oplus c)$$

$$f_2 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b)(a \oplus c \oplus 1)$$

$$f_3 = (a \oplus b \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(a \oplus c)$$

The blue and purple boxes in Figures 30 and 31 show how to convert the POE terms with the same support family from one to another, this process is based on the following properties:

- $A \oplus AB = A\bar{B}$
- $A \oplus B \oplus AB = \bar{A}\bar{B} \oplus 1$

The blue box shows that $(a \oplus b)(a \oplus c \oplus 1)$ can be created by EXOR'ing $(a \oplus b)(a \oplus c)$ with $(a \oplus b)$. The green box shows that $(a \oplus b \oplus 1)(a \oplus c)$ can be created by EXOR'ing $(a \oplus b \oplus 1)(a \oplus c)$ with $(a \oplus c)$.

❖ Using the Phase 2's example 2 expressions, we have:

$$f_1 = (a \oplus c) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus c \oplus d \oplus 1)(c)$$

$$f_2 = (a \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)$$

$$f_3 = (a \oplus c \oplus 1) \oplus (a \oplus d \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1) \oplus (a \oplus d \oplus 1)(b) \oplus (a \oplus 1)(d \oplus 1)(b)$$

$$f_4 = (a \oplus c \oplus 1) \oplus (a \oplus d \oplus 1) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus d \oplus 1)(b) \oplus (a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$$

- Simplifying these expressions we got new expressions:

$$f_1 = (a \oplus c) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus c \oplus d \oplus 1)(c)$$

$$f_2 = (a \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)$$

$$f_3 = (a \oplus c \oplus 1) \oplus (a \oplus d \oplus 1)(b \oplus 1) \oplus (a \oplus 1)(d \oplus 1)(b)$$

$$f_4 = (a \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus d \oplus 1)(b \oplus 1) \oplus (a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$$

- The initial POE terms list is shown in Figure 32

POE terms:	f_1	f_2	f_3	f_4
$(a \oplus c)$	1	1	1	1
$(a \oplus b \oplus 1)(c \oplus 1)$	1	1	0	1
$(a \oplus d \oplus 1)(b \oplus 1)$	0	0	1	1
$(a \oplus c \oplus d \oplus 1)(c)$	1	0	0	0
$(a \oplus 1)(d \oplus 1)(b)$	0	0	1	0
$(a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$	0	0	0	1

Figure 32. Initial POE term list

Among all the POE terms, $(a \oplus c \oplus d \oplus 1)(c)$, $(a \oplus 1)(d \oplus 1)(b)$ and $(a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$ are ungrouped since the number of 1s in its output portion is 1 and so it is therefore separated from the POE-list. The resulting lists are shown in Figure 33 (a).

In the modified POE-list, $(a \oplus c)$ now has the highest number of 1s in its output part. It is thus moved to the sub-list₁. Next, from the remaining POE terms $((a \oplus b \oplus 1)(c \oplus 1)$ and $(a \oplus d \oplus 1)(b \oplus 1))$, a term is selected whose output portion contains the highest number of 1s. So the term $(a \oplus b \oplus 1)(c \oplus 1)$ is moved to sub-list₁. Next the remaining term $(a \oplus d \oplus 1)(b \oplus 1)$ is not allowed to move to sub-list₁ since it is shared by output f_3 which does not contain the term $(a \oplus b \oplus 1)(c \oplus 1)$. There are no other

terms which can be moved to sub-list₁. Figure 33(b) shows the terms in sub-list₁ and POE-list.

Now only one term $(a \oplus d \oplus 1)(b \oplus 1)$ remains in the *POE-list*; thus in the next iteration this term is moved to *sub-list*₂ shown in Figure 33(c), which completes the Phase 3.

POE terms:	$f_1 f_2 f_3 f_4$	POE terms:	$f_1 f_2 f_3 f_4$
$(a \oplus c \oplus d \oplus 1)(c)$	1 0 0 0	$(a \oplus c)$	1 1 1 1
$(a \oplus 1)(d \oplus 1)(b)$	0 0 1 0	$(a \oplus b \oplus 1)(c \oplus 1)$	1 1 0 1
$(a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$	0 0 0 1	$(a \oplus d \oplus 1)(b \oplus 1)$	0 0 1 1
Ungrouped-list		Current POE-list	

(a) Separation of ungrouped POEs from the POE-list

POE terms:	$f_1 f_2 f_3 f_4$	POE terms:	$f_1 f_2 f_3 f_4$
$(a \oplus c)$	1 1 1 1	$(a \oplus d \oplus 1)(b \oplus 1)$	0 0 1 1
$(a \oplus b \oplus 1)(c \oplus 1)$	1 1 0 1	Current POE-list	
Sub-list ₁			

(b) Generation of Sub-list₁

POE terms:	$f_1 f_2 f_3 f_4$
$(a \oplus d \oplus 1)(b \oplus 1)$	0 0 1 1
Sub-list ₂	

(c) Generation of Sub-list₂

Figure 33. POE-list and its sub-lists

The circuit realizations of Examples 2 is shown in Figure 34 with the cost of 83.

Figure 34. Circuit realization of Example 2 produced by EPOEM-MO-1 algorithm with shared-POE-term phase

$$f_1 = (a \oplus c) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus c \oplus d \oplus 1)(c)$$

$$f_2 = (a \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(c \oplus 1)$$

$$f_3 = (a \oplus c \oplus 1) \oplus (a \oplus d \oplus 1)(b \oplus 1) \oplus (a \oplus 1)(d \oplus 1)(b)$$

$$f_4 = (a \oplus c \oplus 1) \oplus (a \oplus b \oplus 1)(c) \oplus (a \oplus d \oplus 1)(b \oplus 1) \oplus (a \oplus b \oplus c \oplus 1)(b \oplus d \oplus 1)(b)$$

5.1.4 Experimental Results:

EPOEM-MO-1 program have been implemented in Python and tested extensively on Unix and Windows workstations. The experimental results presented below have been received on a 2.9 GHz Intel Core i7 PC under Microsoft Windows 8.1.

To verify and compare EPOEM-MO-1 algorithms, several multiple-output benchmark functions were used for testing the synthesis results. The proposed method is compared with several recent works, namely, [5], [83], [84], [85]. Comparisons have been made using 8 benchmarks, out of which EPOEM-MO-1 method provides best result for 6 of them as shown in Table 10. Some of the results of other methods are not shown in the table because they are not provided in their published papers. The results show that the proposed technique results in a very significant reduction in quantum cost. On the other hand, I have also done statistical analysis based on randomization and I tested my algorithm on some random benchmark functions as shown in Figure 35.

Table 10. Experimental results for quantum cost comparison

Function	In	Out	Quantum Cost				
			EPOEM-MO-1	FTR-07 [5]	[83]	[84]	[85]
3_17_6	3	3	29		26	27	
4mod7	4	3	100*		167	110	
f2	4	4	113*	255	246	160	116
4_49_7	4	4	119*		201	174	
aj-e1_81	4	4	71*		201	167	
c17	5	2	91	99	81	85	
cm82a	5	3	46*	154	143	103	69
rd53	5	3	129*	265	269	200	136

Figure 35. Frequency distribution of quantum cost results from 50 random runs using EPOEM-MO-1 to synthesize functions: a) Perk01, b) rd53, c) cm82a

As we can see, to synthesize a multi-output function with n inputs and m outputs, EPOEM-MO-1 transforms this function to a single-output function with $n+m$ inputs. But the current method used to synthesize an incompletely specified single output function is based on template matching method which is limited to 9 variables. In other words, EPOEM-MO-1 can only synthesize a multiple output function with n inputs and m outputs for $(n + m < 9)$. And this is the disadvantage of EPOEM-MO-1 method when compared to other methods.

5.2 EPOEM-MO-2:

5.2.1 Synthesis based on Exclusive-OR Lattices:

The nature of an EXOR logic gate gives the following useful property: given a set of three logic functions A, B, and C, such that $A \oplus B = C$, it is obvious that $B \oplus C = A$ and $A \oplus C = B$. Therefore, given any two functions, the third of this closed set of functions can be uniquely determined. The presented below algorithm exploits this property by performing a logical EXOR between all output functions to search for functions that are commonly repeated or easily implemented. Furthermore, the property scales to any number of functions, introducing a new concept of an EXOR lattice, as shown in Figure 36

In the method presented here, garbage bits are considered to be non-restrictive, and are introduced freely to reduce the complexity of synthesis. Future quantum computers (such as those used for integer factorization) will be useful only for a large number of lines (qubits) [90].

Figure 36. Flattened EXOR Tree (Special case of EXOR Lattice)

The top row of nodes in Figure 36 represents outputs of 6 different functions. Each subsequent row represents the logic functions obtained by logically EXOR'ing the previous row's functions with one another. Therefore, by selecting functions A, F11, and F21, we can implement B directly with a logical EXOR of F11 with A. Performing an EXOR of

F11 and F21 we obtain F12, which can be used to obtain output C, as well. In this case, this map is 2-dimensional (planar). However, when creating such a DAG structure of EXORs, the first and the third node may have some logical EXOR that can be used instead to obtain the third node, rather than having to implement the second to implement the third. This would result in a multi-dimensional DAG (which is a lattice in which the nodes are EXORs of functions being all subsets of the set $\{A,B,C,D,E,F\}$).

An example of the implementation is given in Figures 37, 38, 39 and 40. Figure 37 is the set of Karnaugh maps of desired output functions A, B, and C. Figure 38 shows the logical functions in their EXOR Lattice form.

By implementing functions F1 and A, B can be realized by $F1 \oplus A$. Furthermore, $F3 \oplus F1$ gives a function F2, such that $F2 \oplus B$ realizes C. Therefore, by implementing A, F1, and F3, functions A, B and C can be realized through a series of EXORs. Notice that F1 and F3 are both rather easily implemented, whereas implementation of B and C would have been more costly. Coverings are first found for A, F1, and F3 by using an EXOR synthesis tool such as EXORCISM-4. Such coverings are shown in Fig. 38.

As earlier, we denote by x' the negation of variable x. The A is given by $a \oplus a'bc$, implementation is straightforward, and given in Fig. 39a. Figs 39(b, c) implement F1 and F3, respectively. These can be trivially cascaded, as shown in Fig. 39(d). To implement the remainder of the lattice, the lowest row is implemented first, as in Fig. 40(a), where F2 is implemented from an EXOR of F1 and F3. As functions A, F1, and F2 now exist, the next rows to be implemented are B and C, which are implemented in Figs. 40(b) and 40(c). The complete cascade is shown as Fig. 41.

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

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

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

Figure 37. Karnaugh map that illustrate output function A, B and C

Figure 38. Karnaugh map that illustrate logical functions in their EXOR Lattice form and Coverings are first found for A' , $F1$, and $F3$ by using EXORCISM-4

Figure 39. Circuit realization of: a) Function A, b) Function F1, c) Function F2 and d) Functions: A, F1 and F3 in cascade.

Figure 40. Circuit realization of: a) Function F2 from function F1 and function F3, b) Function B from function A and function F1, c) Function C from function B and function F2

Figure 41. The complete circuit realization of A, B and C

5.2.2 EPOEM-MO-2 Algorithm:

The EPOEM-MO-2 Algorithm which is a combination of EPOEM-1s (or EPOEM-1-DC) with EXOR-lattice concept for the synthesis of completely specified multiple output Boolean functions is shown below. An example of EPOEM-MO-2 synthesis is presented subsequently.

A. Algorithm:

EPOEM-MO-2 algorithm for m-input n-output Boolean function:

Stage 1: Create the EXOR lattice for n given output functions ($A, B, C, \dots, A \oplus B, A \oplus C, \dots$)

Stage 2: Apply EPOEM-1s for each function in the EXOR lattice to find the EPOE expression and its quantum cost.

Stage 3: Select n nodes which have lowest cost to realize n given output functions

The time complexity for stage 1 is $O(C_2^m)$ to create the EXOR-lattice for m output function, for stage 2 is $O(N)$ with N is the number of templates in the library because it uses EPOEM-1-s algorithm and for stage 3 is $O(m \cdot N)$, with m is the number of output functions which is small compare to N, the number of templates in the library, so overall it is $O(N)$. So, the time complexity for EPOEM-MO-2 algorithm is $O(N)$ with N is the number of templates in the library

B. Example: EPOE synthesis for EPOEM-MO-2 algorithm

Below a complete EPOEM-MO-2 synthesis will be performed on a function as shown in Figure 42.

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

A

ab \ cd	00	01	11	10
00		1	1	
01		1	1	1
11	1	1		
10	1			

B

ab \ cd	00	01	11	10
00	1	1	1	1
01		1		
11				
10	1	1	1	

C

ab \ cd	00	01	11	10
00		1		1
01		1		1
11		1	1	1
10			1	

D

ab \ cd	00	01	11	10
00		1	1	
01			1	
11		1	1	
10		1	1	1

Figure 42. Example function that used for EPOEM-MO-1 synthesis

Stage 1: Create the EXOR lattice for n given output functions ($A, B, C, \dots, A \oplus B, A \oplus C, \dots$) as shown in Figure 43.

Figure 43. EXOR-lattice of function given in Figure 42.

Stage 2: Apply EPOEM-1s for each of the function in the EXOR lattice to find the EPOE expression and its quantum cost as shown in Table 11.

Table 11. EPOE expression and quantum cost of each function in EXOR-lattice and the function $(A \oplus D)$

Node	Cost	EPOE expressions
A	33	$(a \oplus c) \oplus (c \oplus d)(a \oplus b \oplus c) \oplus (a \oplus d)(b \oplus d)(c \oplus d)$
B	22	$(b \oplus 1) \oplus (a \oplus d)(b \oplus c)(c \oplus d)$
C	31	$(c \oplus d) \oplus a(b \oplus 1) \oplus a(b \oplus c \oplus 1)(b \oplus d \oplus 1)$
D	20	$d \oplus (a \oplus d)(b \oplus c)(c \oplus d)$
$F11 = A \oplus B$	16	$(a \oplus b \oplus c \oplus 1) \oplus (c \oplus d)(b \oplus 1)$
$F21 = A \oplus C$	33	$(c \oplus d \oplus 1) \oplus (a \oplus c \oplus 1)(d \oplus 1) \oplus (b \oplus 1)c(d \oplus 1)$
$F12 = B \oplus C$	31	$(b \oplus c \oplus 1) \oplus (a \oplus b \oplus c \oplus 1)d \oplus (b \oplus 1)cd$
$F13 = C \oplus D$	21	$c \oplus (a \oplus b)d \oplus bcd$
$F22 = B \oplus D$	5	$(b \oplus d \oplus 1)$
$F31 = A \oplus B \oplus C \oplus D$	25	$(a \oplus 1) \oplus a(d \oplus 1) \oplus bc(d \oplus 1)$
$A \oplus D$	8	$a \oplus b(c \oplus d)$

Stage 3: Select n functions which have the lowest costs to realize n given output functions

Table 12. Sorted functions list base on the quantum costs of the functions

Node	Cost
$F22 = B \oplus D$	5
$A \oplus D$	8
$F11 = A \oplus B$	16
D	20
$F13 = C \oplus D$	21
B	22
$F31 = A \oplus B \oplus C \oplus D$	25
C	31
$F12 = B \oplus C$	31
A	33
$F21 = A \oplus C$	33

Case 1: If we do not use the function $(A \oplus D)$ for this stage:

- From the four single functions (A, B, C, D) in Table 12, function D has the lowest cost so it is selected.

- Based on Table 12, function D exists so function B can be realized by using function $F22 = (B \oplus D)$. The function $F22$ has the cost lower than the function B , so function $F22$ is selected.
- Function B now exists by using node $F22 = (B \oplus D)$, so function $F11 = (A \oplus B)$ is selected to realize function A because it has a lower cost than function A
- Function D exists, so function $F13 = (C \oplus D)$ is selected to realize function C because it has a lower cost than each of the functions C , $F21 = (A \oplus C)$, $F12 = (B \oplus C)$ and $F31 = (A \oplus B \oplus C \oplus D)$
- So, the four functions which have the lowest quantum costs and are used to realize output functions A, B, C and D are: D, F_{11}, F_{13} and F_{22} .

$D = d \oplus (a \oplus d)(b \oplus c)(c \oplus d)$ $F_{11} = (a \oplus b \oplus c \oplus 1) \oplus (c \oplus d)(b \oplus 1)$ $F_{13} = c \oplus (a \oplus b)d \oplus bcd$ $F_{22} = (b \oplus d \oplus 1)$		$A = F_{11} \oplus B$ $B = F_{22} \oplus D$ $C = F_{13} \oplus D$ $D = d \oplus (a \oplus d)(b \oplus c)(c \oplus d)$
---	--	---

The corresponding circuit is given in Figure 44 with quantum cost of 65

Figure 44. Circuit realization of function given in Figure 42 in the case of not using the function $(A \oplus D)$

Case 2: If we use the function $(A \oplus D)$ for this stage:

- From the four output functions (A, B, C, D) , function D has the lowest cost so it is selected.
- Based on Table 12, function D exists so function B can be realized by using function $F22 = (B \oplus D)$, where the function $F22$ has a cost lower than function B , so function $F22$ is selected.
- Function D exists, so function A can be realized by using function $(A \oplus D)$ and function $(A \oplus D)$ has lower cost than function A , so function $(A \oplus D)$ is selected
- Function D exists, so function $F13 = (C \oplus D)$ is selected to realize function C because it has the lower cost than functions C , $F21 = (A \oplus C)$, $F12 = (B \oplus C)$ and $F31 = (A \oplus B \oplus C \oplus D)$
- So, the four functions which have the lowest quantum cost and which are used to realize outputs A, B, C and D are: D , $(A \oplus D)$, F_{13} and F_{22} .

$D = d \oplus (a \oplus d)(b \oplus c)(c \oplus d)$ $(A \oplus D) = a \oplus b(c \oplus d)$ $F_{13} = c \oplus (a \oplus b)d \oplus bcd$ $F_{22} = (b \oplus d \oplus 1)$		$A = (A \oplus D) \oplus D$ $B = F_{22} \oplus D$ $C = F_{13} \oplus D$ $D = d \oplus (a \oplus d)(b \oplus c)(c \oplus d)$
---	--	---

The corresponding circuit is given in Figure 45 with quantum cost of 57

Figure 45. Circuit realization of function given in Figure 42 in the case of using function $(A \oplus D)$

As we can see, using function $(A \oplus D)$ can save the cost by about 12.3%.

I have tested several benchmarks, comparing the quantum cost of the resulting synthesized circuits for n-input and m-output functions between three variants of the EPOEM-MO-2 algorithm, which use different methods to create a subsets of the entire search-space of all logical functions within all EXOR-lattices, as follows:

- (1) Create all the logical functions of the EXOR tree as shown in Figure 36
- (2) Create all the logical functions that EXORs of functions of combination of 2^i ($0 \leq i \leq \log_2 m$) functions of the output set $\{f_1, f_2, \dots, f_m\}$
- (3) Create all the logical functions that are EXORs of functions being all subsets of the output set $\{f_1, f_2, \dots, f_m\}$

Table 13. Quantum cost comparison between the three methods of creating EXOR lattice

Function	In	Out	Quantum Cost		
			(1)	(2)	(3)
3_17_6	3	3	25*	25*	25*
Perk01	4	4	63	57*	57*
Perk02	4	4	55	55	54*
wim	4	7	151	141*	141*
hwb5	5	5	354	346	343
sqr6	6	12	731	711*	711*

Based on comparisons as shown in Table 13, I have chosen the second variant to implement because it almost always gives the same cost as the third variant, and is better than the first variant.

5.2.3 Experimental Results:

EPOEM-MO-2 program have been implemented in Python and tested extensively on Unix and Windows workstations. The experimental results below were obtained on a 2.9 GHz Intel Core i7 PC under Microsoft Windows 8.1.

To verify and compare EPOEM-MO-2 algorithm, several multiple-output benchmark functions were taken from Revlib's page [44] and Maslov's page [45]. The proposed method is compared with several recent works, namely, [5], [82], [83], [84], [85].

Comparisons have been made using 32 benchmarks, out of which EPOEM-MO-2 method provides better result for 28 of them as shown in Table 14. Some of the results of other methods are not shown in the table because they are not provided in their published paper. The results show that the proposed technique results in a very significant reduction in quantum cost. On the other hand, I have also done statistical analysis based on randomization and I tested my algorithm on some random benchmark functions as shown in Figure 46.

a) Function Perk01 with best quantum cost of 54

b) Function rd53 with best quantum cost of 103

c) Function cm82a with best cost of 48

d) Function rd73 with best cost of 276

Figure 46. Frequency distribution of quantum cost results from 50 random runs using EPOEM-MO-2 to synthesize functions: a) Perk01, b) rd53, c) cm82a and d) rd73

Table 14. Experimental results for quantum cost comparison

Function	In	Out	Quantum Cost					
			EPOEM-MO-2	FTR-07 [5]	SD-10 [82]	[83]	[84]	[85]
3_17_6	3	3	25*			26	27	
ex1_82	3	3	15*					
4mod7	4	3	80*			167	110	
f2	4	4	77*	255		246	160	116
4_49_7	4	4	90*			201	174	
aj-e1_81	4	4	50*			201	167	
hwb4	4	4	62					
wim	4	7	141*	217	211	218	172	150
dc1	4	7	185*	416		454	241	
cm42a	4	10	152*	377				
pm1	4	10	152*	377		270		290
c17	5	2	87	99		81	85	
cm82a	5	3	46*	154		143	103	69
rd53	5	3	103*	265	219	269	200	136
hwb5	5	5	346					
squar5	5	8	235*	442		465	393	
c7552_119	5	16	319*	1728		942	992	
decod	5	16	545*	1728		673		976
sqr6	6	12	711*	1033	744			
hwb6	6	6	994					
con1	7	2	175	206		206	171	
rd73	7	3	280*	1143	836	1150	1022	
sqn	7	3	401*	2122				1183
z4	7	4	118*	642		642	448	260
z4ml	7	4	118*	642	577	642	448	260
ham7	7	7	70					
hwb7	7	7	2740					
inc	7	9	1700*	2140				
5xp1	7	10	1144	1430	998	1165		
rd84	8	4	1686*	2749		2558	2477	
sqr8	8	4	704	622		616	604	
radd	8	5	247*	676		669	618	349
adr4	8	5	240*	727		618	618	
dist	8	5	5075*	7601				
root	8	5	2601*	3443				
dc2	8	7	2031	1886				
misex1	8	7	550*	982		1012	725	
hwb8	8	8	3744					
mlp4	8	8	2932*	3753				
urf2	8	8	10363					

*symbols represent those cases where we got better results with respect to other approaches

5.3 Discussion about EPOEM-MO-1 and EPOEM-MO-2 methods:

As shown in Table 10 and 14, from the Experimental Results of EPOEM-MO-1 and EPOEM-MO-2 (Section 5.1.4 and 5.2.3), EPOEM-MO-2 does give a better solution than EPOEM-MO-1. However, for most of the benchmark functions, the results of both EPOEM-MO-1 and EPOEM-MO-2 are better than all other recent ESOP-based synthesis methods' results. Another disadvantage of EPOEM-MO-1 compared to EPOEM-MO-2 is when one attempts to synthesize a multi-output function with n inputs and m outputs. EPOEM-MO-1 transforms this function to a single-output function with $n+m$ inputs. Unfortunately, the current method used to synthesize an incompletely specified single-output function is based on the template matching method which is limited to 9 variables. In other words, EPOEM-MO-1 can only synthesize a multiple output function with n inputs and m outputs for $(n + m < 9)$. In contrast, EPOEM-MO-2 does not have that total numbers of input and output limit, it only has the limit of the number of inputs (limited to 9 inputs when using EPOEM-1 and limited to 16 inputs when using EPOEM-2). The only advantage of EPOEM-MO-1 when compared to EPOEM-MO-2 is that EPOEM-MO-1 can synthesize an incompletely specified output function while EPOEM-MO-2 cannot except for the case that all of the output functions have the same “don't care” set. Concluding, compared to EPOEM-MO-1, the algorithm EPOEM-MO-2 has a better potential for improvement and may lead to deep research as a future work.

Chapter 6: APPLICATIONS OF SYNTHESIS OF EPOE CIRCUITS, QUANTUM AUTOMATA AND REVERSIBLE HARDWARE CLASSIFIERS

Incomplete specifications of single-output Boolean functions have several important applications. First, when the “Muller Method” from Miller’s book [78] is used to synthesize a multi-output function with n inputs and m outputs, this function is transformed to a single-output function with $n+m$ inputs and with a very high percent of don’t cares as already presented in Section 5.1. Next a single-output function is minimized and after minimization shared sub-functions are created based on the inverse Muller Transform [78]. This representation is useful when the DC-set is not represented explicitly, the case in our approach. We use only the ON and the OFF set, which does not increase the number of terms when transforming m -output to 1-output function as shown in Section 5.1. This is a principle “*do not care about don’t cares*” in which don’t cares are represented only implicitly in the algorithm.

Reversible binary automata are a special case of the general concept of quantum automata. The relation of reversible automata to quantum automata is similar to the relation of reversible combinational functions to permutative quantum combinational functions discussed already, so we will not repeat on this subject. Reversible automata are automata composed of combinational logic and flip-flops, in which combinational logic is a reversible circuit. Flip-flops can be integrated to reversible logic or built separately. This reversible circuit may have ancilla bits or not, which leads to two classes of reversible automata. When the reversible combination circuit is realized with quantum gates, superposition and entanglement are possible which leads to the concept of quantum automata. Quantum automaton is realized when inputs to the automaton are in Hilbert

Space or when the automaton includes some non-permutative gates such as Hadamard in addition to the permutative part discussed here. States of quantum automata can be in general not only permutative states but also superposed and entangled states. However in this dissertation we are interested only in automata that have states corresponding to basic quantum vectors of Hilbert state, in other words, with permutative states. Therefore, this category of quantum automata, from the point of logic synthesis, is very similar to quantum circuits as discussed earlier in this dissertation. The only differences are different types of specifications and a potentially high percent of don't cares. We treat quantum permutative automata as a good source of practical examples, benchmarks, for our method, especially that very little has been already published on realization of such automata even with ESOP circuits.

Quantum automata of this type can be realized with any type of reversible logic structures [89], and here it is investigated for the first time, what would be the advantage of realizing them with EPOE. This requires an efficient method of calculating and minimizing multi-output excitation and output functions for incompletely specified automata with arbitrary state and output encoding, which is another reason for developing a don't care based algorithm. Finally, many real-life machines lead to excitation and output functions with very many don't cares because of the following properties: (a) number of states is other than 2^k states which causes functions for all other states encoded by don't cares, (b) there are many don't cares in original transition and output functions, (c) some very good encodings (such as one-hot codes from Table 15 below) with non-minimal codes exist that have highly non-minimal numbers of qubits.

Let us first illustrate how a quantum automaton is realized with an EPOE circuit. Given is an example system, a standard state transition and output table, as shown in Table 14.

Table 15. Example quantum automation for synthesis with EPOEM-MO-2

Present state	Next state				Output
	00	01	10	11	
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S3	0
S2	S1	S3	S2	S0	1
S3	S1	S0	S0	S3	0

- Given the state encoding table as in Table 15

Table 16. States encoding with 1-out-of-4 code

State	Encode
S0	1000
S1	0100
S2	0010
S3	0001

- Replacing the internal states in Table 14 with codes defined in Table 15 gives the new Table 16

Table 17. Example system from Table 14 with encoded states

Present state	Next state				Output
	00	01	10	11	
1000	1000	0100	0010	0001	1
0100	1000	0001	0100	0001	0
0010	0100	0001	0010	1000	1
0001	0100	1000	1000	0001	0

- Apply EPOEM-MO-2 algorithm to the function (6-inputs 5-outputs) in Table 17

Table 18. Example system for synthesis with encoded states

In		Out	
Present state (abcd)	Input (ef)	Next State (ABCD)	Output (E)
1000	00	1000	1
1000	01	0100	1
1000	10	0010	1
1000	11	0001	1
0100	00	1000	0
0100	01	0001	0
0100	10	0100	0
0100	11	0001	0
0010	00	0100	1
0010	01	0001	1
0010	10	0010	1
0010	11	1000	1
0001	00	0100	0
0001	01	1000	0
0001	10	1000	0
0001	11	0001	0

In		Out	
Present state (abcd)	Input (ef)	Next State (ABCD)	Output (E)
0000	--	----	-
11--	--	----	-
1-1-	--	----	-
1--1	--	----	-
-11-	--	----	-
-1-1	--	----	-
--11	--	----	-
111-	--	----	-
1-11	--	----	-
11-1	--	----	-
-111	--	----	-
1111	--	----	-

EPOEM-MO-2 can be applied to this system because all of the output functions share the same “don’t care” set, so we do not need to worry about the case of EXORing between “1-minterm” with a “don’t care term” when we create the logical function in the EXOR-lattice. For example shown in Figure 47, we cannot determine the value in the “??” box is ‘1’ or ‘X’ (don’t care)

A

<div style="display: inline-block; transform: rotate(-45deg); font-size: 0.8em;">c \ ab</div>	00	01	11	10
0	1	X	X	0
1	1	X	X	0

⊕

B

<div style="display: inline-block; transform: rotate(-45deg); font-size: 0.8em;">c \ ab</div>	00	01	11	10
0	1	1	X	1
1	0	X	X	1

=

$A \oplus B$

<div style="display: inline-block; transform: rotate(-45deg); font-size: 0.8em;">c \ ab</div>	00	01	11	10
0	0	??	X	1
1	1	X	X	1

Figure 47. Karnaugh maps that illustrate the the case of EXORing between “1-minterm” with a “don’t care term” when EXORing function A and function B

- Lists of the EPOE expressions and quantum costs of each function after applying the first two stages of EPOEM-MO-2 are shown in Table 18.

Table 19. List of EPOE expressions and their quantum costs of each node

Node	Cost	EPOE expressions
A	49	$(d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus e \oplus 1) (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (e \oplus 1) (d \oplus f \oplus 1)$
B	13	$(b \oplus e \oplus 1) (a \oplus f \oplus 1)$
C	25	$(b \oplus d \oplus 1) (a \oplus c \oplus e \oplus 1) (f \oplus 1)$
D	57	$(a \oplus b \oplus c \oplus d \oplus f \oplus 1) \oplus (b \oplus c \oplus e \oplus 1) (a \oplus d \oplus e \oplus f \oplus 1) \oplus (a \oplus c \oplus d \oplus 1) (b \oplus e \oplus 1) (b \oplus f \oplus 1)$
E	5	$(b \oplus d \oplus 1)$
$F_{12} (A \oplus B)$	77	$1 \oplus (a \oplus b \oplus c \oplus d \oplus f \oplus 1) \oplus (a \oplus c \oplus e \oplus 1) (b \oplus c \oplus d \oplus f \oplus 1) \oplus (b \oplus c \oplus d \oplus 1) (a \oplus b \oplus d \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (d \oplus e \oplus 1) (d \oplus f \oplus 1)$
$F_{13} (A \oplus C)$	60	$(a \oplus d \oplus e \oplus f \oplus 1) \oplus (c \oplus d \oplus 1) (b \oplus e \oplus 1) \oplus (a \oplus b \oplus d \oplus 1) (f \oplus 1) \oplus (a \oplus c \oplus d \oplus 1) (b \oplus e \oplus 1) (b \oplus f \oplus 1)$
$F_{14} (A \oplus D)$	42	$1 \oplus (f \oplus 1) \oplus (c \oplus 1) (d \oplus e \oplus 1) \oplus (a \oplus c \oplus 1) (d \oplus e \oplus 1) (b \oplus d \oplus f \oplus 1)$
$F_{15} (A \oplus E)$	50	$1 \oplus (b \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus f \oplus 1) (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (d \oplus e \oplus 1) (f \oplus 1)$
$F_{23} (B \oplus C)$	43	$(a \oplus f \oplus 1) \oplus (b \oplus c \oplus d \oplus 1) (a \oplus e \oplus 1) \oplus (a \oplus c \oplus 1) (d \oplus e \oplus 1) (f \oplus 1)$
$F_{24} (B \oplus D)$	73	$1 \oplus (a \oplus c \oplus d \oplus e \oplus 1) \oplus (a \oplus e \oplus 1) (a \oplus b \oplus c \oplus d \oplus f \oplus 1) \oplus (b \oplus d \oplus e \oplus 1) (c \oplus e \oplus f \oplus 1) \oplus (a \oplus c \oplus d \oplus 1) (b \oplus e \oplus 1) (f \oplus 1)$
$F_{25} (B \oplus E)$	30	$(b \oplus c \oplus e \oplus f \oplus 1) \oplus (a \oplus c \oplus d \oplus e \oplus 1) (b \oplus c \oplus d \oplus f \oplus 1)$
$F_{34} (C \oplus D)$	60	$(b \oplus d \oplus f \oplus 1) \oplus (b \oplus 1) (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus d \oplus 1) (c \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (d \oplus e \oplus 1) (f \oplus 1)$
$F_{35} (C \oplus E)$	30	$(b \oplus d \oplus 1) \oplus (b \oplus d \oplus 1) (a \oplus c \oplus e \oplus 1) (f \oplus 1)$
$F_{45} (D \oplus E)$	52	$1 \oplus (a \oplus c \oplus f \oplus 1) \oplus (b \oplus c \oplus e \oplus 1) (a \oplus c \oplus d \oplus f \oplus 1) \oplus (a \oplus c \oplus d \oplus 1) (f \oplus 1) (b \oplus e \oplus 1)$
$F_{1234} (A \oplus B \oplus C \oplus D)$	1	1
$F_{1235} (A \oplus B \oplus C \oplus E)$	53	$(a \oplus c \oplus f \oplus 1) \oplus (b \oplus c \oplus e \oplus 1) (a \oplus d \oplus e \oplus f \oplus 1) \oplus (a \oplus c \oplus d \oplus 1) (b \oplus f \oplus 1) (b \oplus e \oplus 1)$
$F_{1245} (A \oplus B \oplus D \oplus E)$	31	$1 \oplus (b \oplus d \oplus 1) \oplus (b \oplus d \oplus 1) (a \oplus c \oplus e \oplus 1) (f \oplus 1)$
$F_{1345} (A \oplus C \oplus D \oplus E)$	18	$(a \oplus c \oplus 1) \oplus (b \oplus e \oplus 1) (a \oplus f \oplus 1)$
$F_{2345} (B \oplus C \oplus D \oplus E)$	49	$(b \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus e \oplus 1) (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (e \oplus 1) (d \oplus f \oplus 1)$

- Select 5 functions from Table 19 that have lowest cost and can generate A', B', C', D' and E' .

Table 20. Sorted functions list by cost

Node	Cost
F1234 ($A \oplus B \oplus C \oplus D$)	1
E	5
B	13
F1345 ($A \oplus C \oplus D \oplus E$)	18
C	25
F35 ($C \oplus E$)	30
F25 ($B \oplus E$)	30
F1245 ($A \oplus B \oplus D \oplus E$)	31
F14 ($A \oplus D$)	42
F23 ($B \oplus C$)	43
F2345 ($B \oplus C \oplus D \oplus E$)	49
A	49
F15 ($A \oplus E$)	50
F45 ($D \oplus E$)	52
F1235 ($A \oplus B \oplus C \oplus E$)	53
D	57
F13 ($A \oplus C$)	60
F34 ($C \oplus D$)	60
F12 ($A \oplus B$)	77
F24 ($B \oplus D$)	73

- From Table 19, functions B and E have the lowest cost, so functions B and E are selected.
- Function C is selected because it cannot be realized from function F1234, F1345, function B and function E .
- Two more functions have to be selected to generate A and D .
 - To realize function A , function with A needs to be realized, for e.g. F12, F13, F15, F1235 and function A . Function A is selected with the lowest cost of 49. As

function A is selected, function D can be generated by selecting function F_{1234} ($A \oplus B \oplus C \oplus D$) with the cost of 1

- To realize function D , function with D needs to be selected, for e.g. F_{24} , F_{34} , F_{45} , F_{2345} and function D . Function F_{2345} with the lowest cost of 49. As function D can be generated from function F_{2345} , B , C and E , function A can be generated by selecting function F_{1234} ($A \oplus B \oplus C \oplus D$) with the cost of 1.
- The way of selecting function A is chosen because it can save the cost of realizing function D from function F_{2345} (have to select F_{2345} to realize D).
- So, the five functions which have the lowest cost to realize output functions A, B, C, D and E are: A, B, C, E and F_{1234}

$$\begin{aligned}
 A &= (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus e \oplus 1) (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (e \oplus 1) (d \oplus f \oplus 1) \\
 B &= (b \oplus e \oplus 1) (a \oplus f \oplus 1) \\
 C &= (b \oplus d \oplus 1) (a \oplus c \oplus e \oplus 1) (f \oplus 1) \\
 F_{1234} &= 1 \\
 E &= (b \oplus d \oplus 1)
 \end{aligned}$$

$$\begin{aligned}
 A &= (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus e \oplus 1) (d \oplus e \oplus f \oplus 1) \oplus (a \oplus b \oplus c \oplus 1) (e \oplus 1) (d \oplus f \oplus 1) \\
 B &= (b \oplus e \oplus 1) (a \oplus f \oplus 1) \\
 C &= (b \oplus d \oplus 1) (a \oplus c \oplus e \oplus 1) (f \oplus 1) \\
 D &= F_{1234} \oplus A \oplus B \oplus C \\
 E &= (b \oplus d \oplus 1)
 \end{aligned}$$

The corresponding circuit of combinational logic of the quantum automaton from Table 14 is given in Figure 48 with quantum cost of 94. The cost of feedback loop and potential flip-flops in it are not taken into account as we are not interested here in the realization methods for quantum automata [89].

Figure 48. Circuit realization of permutative quantum automaton from Table 14. A feedback loop from outputs A, B, C, D to inputs a, b, c, d, respectively is not shown. This loop includes potentially some flip-flops. Signals e and f are primary inputs and signal E is an output of the machine

Another application of our software is for synthesizing hardware classifiers as reversible circuits from EPOE gates from sets of examples. In [90] a method was discussed to use Exclusive-Sum-of-Products (ESOP) Minimizer to synthesize an ESOP circuit that serves as a hardware classifier for Supervised Machine Learning (ML). Similarly, here EPOE synthesis is used for supervised Machine Learning from a set of positive and negative examples, directly to an EPOE circuit that plays a role of a (reversible, quantum) hardware classifier. Minterms are vectors of values of attribute variables (binary in this case). Positive and negative examples are true and false minterms of a highly incomplete Boolean function representation, respectively. Positive case is one for which we learn a positive decision and negative example is one for which we learn a negative decision (i.e. the value of the (output) decision function). Learning here is a generalization that (quasi)-optimally converts don't cares to cares to satisfy the Occam Razor Principle. In case of our circuits, the Maslov quantum cost is used to represent the Occam Razor when minimizing the cost of the quantum classifier circuit.

Chapter 7: CONCLUSION

This dissertation introduces a new concept of reversible circuits based on EXOR-sum of Products-of-EXOR-sums called EPOE. It must be noted that the EPOE idea is new to reversible logic synthesis. Moreover, the work presented in this dissertation is a pioneering attempt to create algorithms based on the EPOE concept. Several new methods, which are EPOEM-1s, EPOEM-1f, EPOEM-2, EPOEM-1-DC, EPOEM-1-DC-tree, EPOEM-MO-1, EPOEM-MO-2 and EPOE-EXACT for synthesis of reversible circuits with no ancilla bits and with small ancilla bits based on EPOE concept have been developed for each category of: completely specified functions, incompletely specified functions and multiple output functions.

Three new software tools named EPOEM-1s, EPOEM-1f and EPOEM-2 have been introduced in this dissertation, which synthesize arbitrary single-output functions for quantum and reversible circuits with EPOE type circuits. Compared with EXORCISM-4, the common ESOP synthesis tool that has been used for over 10 years, over many benchmark functions, both EPOEM-1s and EPOEM-1f consistently produced solutions of equal or lower quantum cost with improvements ranging up to typically 50%, and in some cases up to 85%. EPOEM-1f always gives a better solution than the EPOEM-1s algorithm but the circuit produced by it has an additional ancilla line for each common POE term. On the other hand, the EPOEM-2 algorithm (using Boolean factorization), cannot produce as good solutions as the EPOEM-1 algorithm, but compared to EXORCISM-4 it produced solutions of equal or lower quantum cost for 41/46 benchmark functions. The main advantages of the EPOEM-2 over EPOEM-1 is that it can synthesize up to 16 variable input

functions and the result is calculated faster. While the limit of EPOEM-1 is that it can synthesize only up to 9 variables.

Moreover, two other software tools, EPOEM-1-DC and EPOEM-1-DC-tree, have been also developed for synthesis of incompletely specified single output functions. The EPOEM-1-DC-tree algorithm always gives the better solution but takes much longer time to generate the result when the function has more than 6 variables compared to the EPOEM-1-DC algorithm.

The EPOEM-MO-1 and EPOEM-MO-2 tools are also developed for synthesis of multiple output functions. Compared with current ESOP-based approach methods like in [5], [82], [83], [84] and [85] as shown in Table 10 and Table 14, both EPOEM-MO-1 and EPOEM-MO-2 mostly produced better solutions for 6/8 and 28/32 benchmark functions respectively, with a very significant reductions in quantum costs. The experiments and analyses presented in this dissertation should lead to an increased interest of EPOE concept in the reversible logic synthesis research community, which now uses predominantly the ESOP-based circuits.

For future work, I plan on improving the EPOEM-2 algorithm so it will be able to optimize the results in order to always produce the solutions of equal or lower quantum cost when compared to EXORCISM-4. I also plan on a post processing algorithm aiming at further reduction of the synthesized circuit through sharing the common POE term in the EPOE expression. I also plan on improving the EPOEM-MO-2 algorithm so it can also synthesize arbitrary incompletely specification multiple output functions which will be used in quantum automata synthesis.

WORKS CITED

- [1] N. Alhagi, "Synthesis of Reversible Functions Using Various Gate Libraries and Design Specifications", Ph.D. Dissertation, Jan 2010, PSU, ECE Dept.
- [2] N. Alhagi, M. Hawash and M. A. Perkowski, "Synthesis of Reversible Circuits with No Ancilla Bits for Large Reversible Functions Specified with Bit Equations", Proc. ISMVL, pp. 39-45 (2010).
- [3] N. Alhagi, M. Lukac, L. Tran and M. Perkowski: "Two-stage Approach to the Minimization of Quantum Circuit Based on ESOP Minimization and Addition of a Single Ancilla Qubit", 21st Int. Workshop on Post-Binary ULSI Systems, 25-36 (2012)
- [4] A. Cheng, E. Tsai, M. Perkowski, A. Rajendar and Y. Wang: "Comparison of Maslov's Quantum Costs and LNNM Quantum Costs for Four Types of Multi-qubit Toffoli Gates", 21st Int. Workshop on Post-Binary ULSI Systems, 81-87 (2012)
- [5] K. Fazel, M. Thornton, and J. E. Rice, "ESOP-based Toffoli gate cascade generation," Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 206-209, August 2007.
- [6] Z. Hamza and G. W. Dueck: "Near-Optimal Ordering of ESOP Cubes for Toffoli Networks", 2nd Workshop on Reversible Computation, 49-53 (2010)
- [7] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis", Proceedings of the Design Automation Conference, p. 318-323, June 2003.
- [8] D. M. Miller, R. Wille, and R. Drechsler, "Reducing Reversible Circuit Cost by Adding Lines", Proc. ISMVL, pp. 217 - 222 (2010)

- [9] A. Mishchenko and M. Perkowski, "Fast Heuristic Minimization of Exclusive Sums-of-Products," Proc. RM 2001 Workshop, p. 242-250, August 2001.
- [10] A. Mishchenko and M. Perkowski, "Logic Synthesis of Reversible Wave Cascades," Proc. IEEE/ACM International Workshop on Logic Synthesis, pp. 197 – 202, June 4-7 (2002)
- [11] N. M. Nayeem, and J. E. Rice, "A Shared-cube Approach to ESOP-based Synthesis of Reversible Logic," Facta Universitatis Series : Electronics and Energetics, vol. 24, no. 3, pp. 385-402, December 2011.
- [12] M. Saeedi, M. S. Zamani, M. Sedighi, and Z. Sasanian. "Reversible circuit synthesis using a cycle-based approach," ACM Journal on Emerging Technologies in Computing Systems, vol. 6, no. 4, pp. 1-26 (2010)
- [13] T. Sasao. Logic Synthesis and Optimization, Kluwer Academic Publishers, Norwell, MA, USA, pp. 269 (1993)
- [14] B. Schaeffer, L. Tran, A. Gronquist, M. Perkowski and P. Kerntopf, "Synthesis of Reversible Circuits Based on Product of Exclusive Or Sums", Proc. ISMVL, pp. 35 – 40, May 2013
- [15] A. De Vos, "Reversible Computing: Fundamentals, Quantum Computing, and Applications", Weinheim: Wiley-VCH (2010)
- [16] R. Wille, and R. Drechsler, "BDD-Based Synthesis of Reversible Logic," Int. J. of Applied Metaheuristic Computing 1(4), pp. 25-41 (2010)
- [17] T. Toffoli, "Reversible computing" . Springer, 632. Technical Memo MIT/LCS/TM-151, MIT Lab. For Comput. Sci (1980)

- [18] M. Saeedi and I. L. Markov, "Synthesis and Optimization of Reversible Circuits - A Survey", *ACM Computing Surveys*, 45, 2, Article 21 (2013)
- [19] S. Lee, S. J. Lee, T. Kim, J. S. Lee, J. Biamonte, and M. Perkowski., "The cost of quantum gate primitives", *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6) (2006)
- [20] V. Shende and I. L. Markov, "On the CNOT-cost of TOFFOLI gates", *Quant. Inf. Comput.* 9, 5-6, pp. 461–486 (2009)
- [21] D. Maslov and M. Saeedi "Reversible circuit optimization via leaving the Boolean domain". *IEEE Trans. CAD* 30, 6, pp. 806 – 816 (2011)
- [22] Y.V. Rentergem, A.D. Vos, and K.D.Keyser, "Six synthesis methods for reversible logic." *Open Sys. & Inf. Dynamics* 14, 1, pp. 91–116 (2007)
- [23] L. Storme, A.D. Vos and G. Jacobs. "Group theoretical aspects of reversible logic gates". *J. of Universal Comput. Sci.* 5, 5, pp. 307–321 (1999)
- [24] G. Yang, X. Song, W.N. Hung, F. Xie and M. Perkowski, "Group theory based synthesis of binary reversible circuits". *Lec. Notes in Comp. Sci.* 3959/2006, 365–374 (2006)
- [25] D. Maslov, "Linear depth stabilizer and quantum Fourier transformation circuits with no auxiliary qubits in finite neighbor quantum architectures". *Phys. Rev. A* 76 (2007)
- [26] R. Wille and R. Dreschsler, "BDD-based synthesis of reversible logic for large functions." *Design Autom. Conf.*, pp. 270–275 (2009)
- [27] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, pp. 183-191 (1961)

- [28] C. Bennett, "Logical Reversibility of computation," *IBM Journal of Research and Development*, pp. 525-532 (1973)
- [29] C. H. Bennett. "Notes on the history of reversible computation", *IBM J. Research and Development*, pages 32(1):16-23, January 1988.
- [30] R. P. Feynman. "Feynman Lectures on Computation", Addison-Wesley, paperback edition, (1996)
- [31] I. Zutic, J. Fabian, and S. Das Sarma. "Spintronics: Fundamentals and applications", *Rev. Modern Phys.*, p. 323-410 (2004)
- [32] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch, "Practical strategies for power-efficient computing technologies", *Proceedings of the IEEE*, p. 215-236 (2010)
- [33] N. Gershenfeld and I. L. Chuang. "Quantum computing with molecules", *Proc. of Scientific American*, June 1998.
- [34] M. Nielsen and I. Chuang. "Quantum Computation and Quantum Information", Cambridge University Press (2000)
- [35] J. Preskill, "Lecture notes in quantum computing", <http://www.theory.caltech.edu/preskill/ph229>
- [36] H. Rahaman, D. K. Koley, D.K. Das, and B. B. Bhattacharya, "Detection of bridging fault in reversible circuit", *Proc. of International Conference on VLSI Design and test Symposium, VDAT2006*, p. 384-392, August 2006.
- [37] P. Picton. "A universal architecture for multiple-valued reversible logic" *Multiple-Valued Logic Journal*, p. 27-37 (2000)
- [38] R. C. Merkle and K. E. Drexler, "Helical logic", *Nanotechnology*, p. 325-339 (1996)

- [39] R. C. Merkle. "Reversible electronic logic using switches", Nanotechnology, p. 21-40 (1993)
- [40] R. C. Merkle. "Two types of mechanical reversible logic", Nanotechnology, p. 114-131 (1993)
- [41] R. Feynman. "Quantum mechanical computers", Optic News, volume 11(2), p. 11-20 (1985)
- [42] M. Perkowski , T. Ross, D. Gadd , J. A. Goldman , N. Song, "Application of ESOP Minimization in Machine Learning And Knowledge Discovery", Proc. RM, Makuhari, Japan (1995)
- [43] A. Alfonso-Faus, "Fundamental Principle of Information-to-Energy Conversion", Plenary Lecture WSEAS, published in "Recent Advances in Information Science", Proc. 7th European Computing Conference (EEC'13), pp. 373-376, Dubrovnik, Croatia, June 25-27, 2013
- [44] RevLib - An Online Resource for Reversible Functions and Circuits. [Online]. http://revlib.org/function_details.php?id=12
- [45] D. Maslov. (2011) Reversible Logic Synthesis Benchmarks Page. [Online]. <http://webhome.cs.uvic.ca/~dmaslov>
- [46] A. K. Prasad, I. L. Markov, and J. P. Hayes V. V. Shende, "Synthesis of reversible logic circuits," IEEE Transactions on CAD, vol. 22, no. 6, pp. 710-722, 2003.
- [47] X. Song, W. N. N. Hung, and M. A. Perkowski G. Yang, "Fast synthesis of exact minimal reversible cricuits using group theory," in ASP Design Automation, Asia and South Pacific, pp. 1002-1005 (2005)

- [48] R. Wille, G. W. Dueck, and R. Drechsler D. Große, "Exact multiple control Toffoli network synthesis with SAT techniques," *IEEE Transaction on CAD*, vol. 28, no. 5, pp. 703-715 (2009)
- [49] N. Alhagi, M. Hawash, and M. Perkowski, "Synthesis of Reversible Circuits with No Ancilla Bits for Large Reversible Functions," in *International Symposium on Multiple-Valued Logic*, Barcelona, Spain, pp. 39-45 (2010).
- [50] M. Soeken, R. Wille, and R. Drechsler, "Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition," in *5th International Design and Test Workshop*, Abu Dhabi, p. 143-148 (2010).
- [51] R. Wille, M. Saeedi, and R. Drechsler, "Synthesis of reversible functions beyond gate count and quantum cost," in *Int't Workshop on Logic Synthesis*, pp. 43-49 (2009)
- [52] V. Shende, S. Bullock, and I. Markov, "Synthesis of Quantum-Logic Circuits," *IEEE Transactions on Computer-Aided design of intergrated circuits and systems*, vol. 25, no. 6, p. 1000 (2006)
- [53] D.M. Miller and M.A. Thornton, "QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits," in *ISMVL*, p. 30 (2006)
- [54] Ch. Stedman, B. Yen, and M. Perkowski, "Synthesis of Reversible Circuits with Small Ancilla Bits for Large Irreversible Incompletely Specified Multi-Output Boolean Functions," in *Proc. 14th International Workshop on Post-Binary ULSI Systems* (2005)
- [55] D. Maslov, C. Young, D.M. Miller, and G.W. Dueck, "Quantum circuit simplification using templates," in *Design, Automation and Test in Europe (DATE)*, Europe, p. 1208 – 1213 (2005)

- [56] M. Hawash, M. Perkowski, S. Bleiler, J. Caughman., and A. Hawash, "Reversible Function Synthesis of Large Reversible Functions With No Ancillary Bits Using Covering Set Partitions," in 8th International Conference on Information Technology-New Generation, Las Vegas, NV, p. 1008 – 1013 (2011)
- [57] B. Desoete and A. D. Vos, "A reversible carry-look-ahead adder using control gates," Integration, the VLSI journal, vol. 33, no. 1-2, pp. 89-104 (2002)
- [58] R. Van Meter and M. Oskin, "Architectural Implications of Quantum Computing Technologies," ACM Journal on Emerging Technologies in Computing Systems, vol. 2, no. 1, pp. 31-63, Jan 2006.
- [59] D. Deutsch, "Quantum theory, the church-turing principle and the universal quantum computer," Royal Society London Series A, vol. 400, p. 97 (1985)
- [60] P.W Shor, "Algorithms for quantum computation: Discrete log and factoring," 35th Annual Symposium on the Foundations of Computer Science, Los Alamitos, CA, p. 124 (1994)
- [61] L. K. Grover, "A fast quantum mechanical algorithm for database search," 28th Annual ACM Symposium on the Theory of Computing, New York, NY, p. 212 – 219 (1996)
- [62] P. W. Shor, "Scheme for reducing decoherence in quantum memory.," Physics Review Letters, no. 52, p. 2493 (1995)
- [63] A. Barenco, "Elementary gates for quantum computation," Physical Review A, vol. 52, pp. 52:3457–3467 (1995)
- [64] A. Younes and J. Miller, "Representation of Boolean Quantum Circuits as Reed Muller Expressions," arxiv: quant-h/0304134, May 2003.

- [65] A. Chakrabarti and S. Sur-Kolay, "Nearest Neighbour based Synthesis of Quantum Boolean Circuits," *Engineering Letters*, vol. 15, no. 2, Nov 2007.
- [66] M. Perkowski, M. Lukac, D. Shah, and M. Kameyama, "Synthesis of quantum circuits in Linear Nearest Neighbor model using Positive Davio Lattices," *Facta Universitatis (NIS)*, vol. 24, no. 1, pp. 73-89, April 2011.
- [67] B. Bollig, M. Lobbing, M. Sauerhoff, and I. Wegener, "On The Complexity of the Hidden Weighted Bit Function for Various BDD Models," *Informatique Theorique et Applications*, vol. 33, no. 2, pp. 103-116 (1999)
- [68] X. Zeng, Marek A. Perkowski, Haomin Wu, and Andisheh Sarabi, "A New Efficient Algorithm for Finding Exact Minimal Generalized Partially-Mixed-Polarity Reed-Muller Expansion," *Proc. of the Second Workshop on Applications of Reed-Muller Expansion in Circuit Design*, Chiba City, Japan, 27-29 August 1995, pp. 231-239.
- [69] X. Zeng, Marek A. Perkowski, Karen Dill, and Andisheh Sarabi, "Approximate Minimization of Generalized Reed-Muller Forms," *Proc. of the Second Workshop on Applications of Reed-Muller Expansion in Circuit Design*, Chiba City, Japan, pp. 221-230, August 1995.
- [70] H. Wu, M. A. Perkowski, N. Zhuang, "Generalized Partially-Mixed-Polarity Reed-Muller Expansion and its Fast Computation," *IEEE Transactions on Computers*, Vol. 45, No. 9, pp. 1084-1088, September 1996.
- [71] M. A. Perkowski, and P. D. Johnson, "Canonical Multivalued-Input Reed-Muller Trees and Forms," *Proc. of the Third NASA Symposium on VLSI Design*, pp. 11.3.1-11.3.13, Moscow, Idaho, October 1991.

- [72] I. Schaefer, and M. A. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms," *Proc. of the 21th IEEE International Symposium on Multiple Valued Logic, ISMVL'91*, pp. 40 - 48, Victoria, British Columbia, May 1991.
- [73] A. Younes and J. Miller, " Representation of Boolean Quantum Circuits as Reed Muller Expressions", arxiv: quant-ph/0304134, May 2003.
- [74] M. A. Harrison, "On the classification of Boolean functions by the general linear and affine groups," *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 2, pp. 285–299 (1964)
- [75] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, H. Weinfurter, "Elementary Gates for Quantum Computation". *Physical Review A*, 52, 3457–3467 (1995)
- [76] D. Maslov, G. W. Dueck, "Improved Quantum Cost of n-bit Toffoli Gates", *Electronics Letters*, 39, 1790–1791 (2003)
- [77] F. Luccio, L. Pagli, "On a new Boolean function with applications", *IEEE Trans. Comput.*, 48, 3, 296-310 (1999)
- [78] R.E. Miller, "Switching Theory", John Wiley, New York (1965)
- [79] V. Ciriani, "Synthesis of SPP Three-Level Logic Networks Using Affine Spaces", *IEEE Transactions on TCAD*, 22(10) 1310–1323 (2003)
- [80] A. Bernasconi, V. Ciriani, R. Drechsler, and T. Villa, "Efficient Minimization of Fully Testable 2-SPP Networks", Technical Report TR-05-23, University of Pisa (2005)

- [81] V. Ciriani, A. Bernasconi, and R. Drechsler, “Testability of SPP Three-Level Logic Networks.”, IFIP 12-th International Conference on Very Large Scale Integration, (VLSISOC), pp. 331–336 (2003)
- [82] Y. Sanaee, G. W. Dueck, “ESOP-based Toffoli network generation with transformations”, Proc. ISMVL, IEEE Computer Society, pp. 276–281 (2010)
- [83] J. E. Rice and N. M. Nayeem, “Ordering techniques for ESOP-based toffoli cascade generation”, PacRim2011, pp. 274 – 279, August 2011.
- [84] C. Bandyopadhyay, D. Roy, D. K. Kole, K. Dutta, H. Rahaman, “ESOP-based synthesis of reversible circuit using improved cube list”, Proc. ISED, pp 26-30 (2013)
- [85] C. Bandyopadhyay, H. Rahaman, R. Drechesler “A Cube Pairing Approach for Synthesis of ESOP-based Reversible Circuit”, IEEE 44th International Symposium on MultipleValued Logic (ISMVL-2014), pp. 109-114 (2014)
- [86] A. Tran, J. Wang: “A Decomposition Method for Minimisation of Reed-Muller Polynomials in Mixed Polarity”, IEE Proc., Computers and Digital Techniques, 140, 65-68 (1993)
- [87] J. Wang: “A decomposition method lor the minimization of exclusive-OR switching functions”. MS thesis, University of Lowell, Lowell, Massachusetts (1991)
- [88] L. Tran, B. Schaeffer, A. Gronquist, M. Perkowski, P. Kerntopf: “Synthesis of Reversible Circuits Based on EXORs of Products of EXORs”, Trans. on Comput. Sci. XXIV, LNCS 8911, pp. 111–128 (2014)
- [89] M. Lukac, M. Kameyama, and M. Perkowski: “Quantum Finite State Machines - a Circuit Based Approach, Quantum Information Processing”, International Journal of Unconventional Computing, Vol. 9, Num. 3-4, pp. 267-301 (2013)

- [90] V.S. Shivgand, A.Aulakh, and M. Perkowski, "Quantum Circuit Layout", Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies, Tokyo, Japan, September 2005.
- [91] M. Perkowski and M. Chrzanowska-Jeske, "An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions," Proc. ISCAS, pp. 1652-1655, June 1990.
- [92] A. Gaidukov, "Algorithm to derive minimum ESOP for 6-variable function", Proc. of the 5th International Workshop on Boolean Problems, Freiberg (2002)
- [93] T. Sasao, "EXMIN2: A simplification algorithm for Exclusive-or-Sum-of-Products expressions for multi-valued input two-valued output function", IEEE Trans. CAD, Vol. 12, No.5, pp. 621-623, May 1993
- [94] N. Song and M. Perkowski, "Minimization of exclusive sum of products expression for multi-output multi-valued input, incompletely specified functions". IEEE Trans. CAD, Vol. 15, No.4, pp. 385-395, April 1996
- [95] D. Voudouris, M. Kalathas, G. Papakonstantinou: "Decomposition of Multi-output Boolean Functions", HERCMA 2005, Athens, Hellas (2005)
- [96] L. Tran, B. Yen, M. Perkowski: "Comparison of Various Error-Detecting And Error-Correcting Encodings of Reversible Automata Built From Irreversible State Tables using EPOE Circuits with EXOR Lattices", Preparing for Submission to a Journal.

Appendix A: Glossary

A

Adiabatic CMOS

Realization of reversible adiabatic circuits using CMOS technology, often in dual rail logic.

Ancilla bits

Additional bits added to a reversible circuit (qubits added to a quantum permutative circuit) to allow realization of irreversible function with reversible gates or to decrease the number of gates in the reversible circuit that realizes a reversible function.

AND/EXOR circuits

Circuits built entirely from gates NOT, AND, EXOR and constant 1. They can have two or more levels.

B

Balanced function

A Boolean function that has the same number of ones and zeros as its outputs (it means, the same number of true and false minterms in the K-Map).

BDD

Binary Decision Diagram. Data structure used in CAD based on Shannon expansions (multiplexers). It is a DAG (Directed Acyclic Graph) which combines all isomorphic nodes (subfunctions) and has control variables ordered.

Bi-directional synthesis algorithm

Search algorithm in which branching is executed from input to output and from output to input at the same time.

C

Circuit Model of quantum computing

This is a classical and historically first and most developed model of quantum computing. It is based on quantum gates used in this dissertation and other gates. Other models of quantum computing include Quantum Turing Machine, Quantum Automata, and Quantum Cellular Automata. More models have been recently introduced such as cluster quantum computing, adiabatic quantum computing, topological quantum computing, etc.

Circuit width

Width of a quantum (reversible) circuit is the number of qubits (in a “quantum register”) or a number of bits (in a reversible circuit). Counting the width we include all bits, ancilla

and garbage bits. Circuit width is some measure of circuit complexity. It is used separately from the quantum cost or the number of (quantum) gates by some authors.

D

Davio Expansion.

Davio Expansion is an expansion of Boolean functions that uses the so-called Davio gate $f = ab \oplus c$ and reduces one variable from the original Boolean expression. It is similar to Shannon Expansion, but while Shannon is used in BDDs, Davio expansions are used in Kronecker Functional Decision Diagrams. There are Positive Davio expansions that use positive polarity variable for expansion and Negative Davio Expansions that use negative polarity variable.

E

ESOP

Exclusive-Or Sum of Products circuits that are a fundament of AND/EXOR circuit synthesis.

Exact synthesis algorithm

Exact synthesis algorithm is an algorithm that guarantees obtaining the minimum correct solution (synthesizing a correct circuit, one that matches the initial specification). Minimum is in the sense of minimizing the cost function. Cost function can be number of gates, number of inputs to gates, total cost of library cells, quantum cost, total delay, etc.

F

Fan-out

Fan-out of a gate G is a number of gates to which the output of the gate G goes. In case of reversible circuits the fan-out of every output is one.

FPRM

Fixed Polarity Reed-Muller canonical forms. FPRM circuits are a type of AND/EXOR circuits which are canonical and can be also synthesized using spectral synthesis methods and algebraic methods. The FPRM forms, especially their special case PPRM (Positive Polarity Reed-Muller Forms) are used in reversible circuit synthesis.

G

Garbage signal (bit, qubit)

Garbage is an output that has no any logical use and it exists in the reversible circuit for the sake of making this circuit reversible (permutative). Garbage waste energy in nonquantum technologies. They waste computing resources in quantum technologies,

hence their name. They waste also energy in quantum computing when they are measured.

Gate cost

Gate cost is the same as the total number of reversible gates in the circuit. Called also “circuit length”. It is an approximate metric used in some synthesis algorithms. Now it is mostly replaced by quantum cost.

Go-through wires

Wires (bits, signals, qubits) that go through a reversible gate from input to output and are not modified.

Grover Algorithm

Grover Algorithm is a famous quantum algorithm invented by Lov Grover from Bell Labs for a standard quantum circuit computer model. This algorithm finds an item in the so-called non-ordered data base reducing time from N to square-root-of- N . Many NP problems can be reduced to Grover, for instance SAT, graph coloring, Boolean minimization, etc. Grover algorithm specifies the problem to be solved by building a logical oracle for it, and the oracle is a reversible (quantum permutative) circuit, which leads to the area of synthesis of such circuits.

Group theory

Mathematical theory about groups, i.e. algebraic structures that satisfy axioms of one operation called group multiplication. Group Theory is used in synthesis of reversible and irreversible logic circuits and quantum circuits.

Group gate

Group gate is a logic gate that satisfies the mathematical axioms of a group. Modulo additions and GF additions are examples of group gates.

H

Hamming Distance

Hamming Distance of two binary vectors is the number of positions in which these vectors differ.

I

Incompletely specified functions.

Incompletely specified functions are Boolean functions with don't cares. For some input combinations the output is arbitrary.

Information Loss.

Bennett and Landauer linked the concepts of information theory (entropy, measures of information) to the energy loss during computer's calculations. They linked information

loss further to the logical design of gates for low power. An example of a circuit that loses information is a two-input AND gate, which produces value 0 on gate's output 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 of the AND gate. According to Bennett and Landauer, it is a necessary condition to use only reversible gates to build a circuit that will not lose energy during (internal) calculations (Energy is, however, lost for input and output operations).

K

Kronecker Functional Decision Diagram (KFDD)

Decision diagram that uses ordered expansion variables and Shannon, Positive Davio or Negative Davio gates (expansions) in each level to expand function F recursively. There are many special forms of KFDDs, such as those that use only Positive Davio expansions and have their variables ordered.

M

Mixed Polarity Circuits

Logic circuits such as ESOP or Generalized Reed Muller in which variables stand in both positive and negative polarities in all product terms.

MMD

MMD is the software for synthesis of reversible circuits developed by Miller, Maslov and Dueck. It has been permanently improved by several teams since 2003.

P

Permutative Circuit, Permutative Quantum Circuit, Reversible circuit

While all quantum circuits are described by unitary matrices, their subset, the permutative circuits (reversible circuits) are described by unitary matrices which correspond to permutations of their rows and columns. These types of matrices are the so-called permutative matrices. A permutative circuit permutes input vectors to output vectors. Such circuits can be described by some type of truth tables.

Q

Quantum cellular automata

Quantum Cellular Automata are circuits built in Quantum Dot or similar quantum technologies. Formally they are cellular automata but they realize Boolean logic with

majority gates and inverters. This is the most advanced quantum technology that allows to build traditional microprocessors.

Quantum circuits

Quantum circuits and gates are those that are described by arbitrary unitary matrices.

Quantum costs

Costs of quantum gates calculated by Soonchill Lee, Maslov and others for every Toffoli gate with n inputs. They are used to calculate costs of quantum permutative circuits. Approximately they grow quadratically with the number of inputs. A standard metric used in synthesis algorithms. There are several variants of costs related to some technologies or calculated in more or less approximate ways for various gate libraries. This dissertation uses the most well-known “Maslov’s costs”.

Quantum Circuit Synthesis.

Synthesis of quantum circuits (discrete in contrast to analog or continuous) that starts from a unitary matrix $\mathbf{u}$ specification ($\mathbf{u} \times \mathbf{u}^+ = \mathbf{I}$) of a circuit and decomposes this initial specification to unitary matrices of realizable “quantum gates” such as Hadamard gates, Feynman or Toffoli gates. In this dissertation, we solve a subset of this problem by assuming that the unitary matrix is permutative. Thus, the corresponding circuit can only include permutative gates such as NOT, Feynman, and Toffoli.

R

Reversible logic operations

Reversible logic operations are certain logic operations that do not erase information . When a computational system erases a bit of information, it dissipates energy of $\log 2 \times KT$ Joule where K is Boltzmann’s constant and T is the temperature. Reducing power is the main task of modern digital circuit design, making design with reversible circuits of interest as it reduces power that is dissipated by computing systems.

Reversible logic synthesis

Reversible logic synthesis is area of logic synthesis which is concerned with synthesis of reversible circuits.

S

Shor Algorithm

Quantum algorithm for factorization of integers used in cryptography. It gives exponential speedup.

T

Template Matching.

Template matching is an approach to local optimization of reversible circuits based on applying templates that are shifted through the reversible circuit to perform local transformations that reduce the quantum cost.

Toffoli gate.

Toffoli gate is the main gate in reversible design as it is universal. It realizes the functions $A = a$, $B = b$, $C = ab \oplus c$. Outputs A and B are thus go-through signals and C realizes a Davio expansion.

U

Unitary matrix

A Unitary matrix is a matrix U of complex numbers such that its matrix product with its hermitian matrix $U^\dagger$ is an identity. Hermitian matrix is a conjugate of a transposed matrix.

W

Wave cascades

Wave Cascades are reversible circuits which are exors of Maitra cascades realized with reversible gates. Maitra Cascades were invented by Maitra but they are not universal. Wave Cascades are universal and were invented by Mishchenko and Perkowski.