

Portland State University

**PDXScholar**

---

Dissertations and Theses

Dissertations and Theses

---

1-1-2010

# Synthesis of Reversible Functions Using Various Gate Libraries and Design Specifications

Nouraddin Alhagi

*Portland State University*

Follow this and additional works at: [https://pdxscholar.library.pdx.edu/open\\_access\\_etds](https://pdxscholar.library.pdx.edu/open_access_etds)

**Let us know how access to this document benefits you.**

---

## Recommended Citation

Alhagi, Nouraddin, "Synthesis of Reversible Functions Using Various Gate Libraries and Design Specifications" (2010). *Dissertations and Theses*. Paper 366.

<https://doi.org/10.15760/etd.366>

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. Please contact us if we can make this document more accessible: [pdxscholar@pdx.edu](mailto:pdxscholar@pdx.edu).

Synthesis of Reversible Functions  
Using Various Gate Libraries  
and Design Specifications

by

Nouraddin Alhagi

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  
Robert Daasch  
Christof Teuscher  
Wayne Wakeland  
Douglas Hall

Portland State University  
2010

## **Abstract**

This dissertation is devoted to efficient automated logic synthesis of reversible circuits using various gate types and initial specifications. These Reversible circuits are of interest to several modern technologies, including Nanotechnology, Quantum computing, Quantum Dot Cellular Automata, Optical computing and low power adiabatic CMOS, but so far the most important practical application of reversible circuits is in quantum computing. Logic synthesis methodologies for reversible circuits are very different than those for classical CMOS or other technologies. The focus of this dissertation is on synthesis of reversible (permutative) binary circuits. It is not related to general unitary circuits that are used in quantum computing and which exhibit quantum mechanical phenomena such as superposition and entanglement. The interest in this dissertation is only in logic synthesis aspects and not in physical (technological) design aspects of reversible circuits. Permutative quantum circuits are important because they include the class of oracles and blocks that are parts of oracles, such as comparators or arithmetic blocks, counters of ones, etc. Every practical quantum algorithm, such as the Grover Algorithm, has many permutative circuits. These circuits are also used in Shor Algorithm (integer factorization), simulation of quantum systems, communication and many other quantum algorithms. Designing permutative circuits is therefore the major engineering task that must be solved to practically realize a quantum algorithm. The dissertation presents the theory that leads to MP (Multi-Path) algorithm, which is currently the top minimizer of reversible circuits with no ancilla bits. Comparison of MP with other leading software tools is done. This software allows to minimize functions of more

variables and with smaller quantum cost than other CAD tools. Other software developed in this dissertation allows to synthesize reversible circuits for functions with “don’t cares” in their initial specifications. Theory to realize functions from relational representations is also given. Our yet other software tool allows to synthesize reversible circuits for new types of reversible logic, for which no algorithm was ever created, using the so-called “pseudo-reversible” gates called Y-switches.

## Table of Contents

Abstract .....	i
List of Figures .....	iv
List of Tables .....	ix
Chapter 1: INTRODUCTION.....	1
Chapter 2: BACKGROUND AND LITERATURE REVIEW ON BASIC REVERSIBLE GATES.....	20
Chapter 3: REALIZATION OF REVERSIBLE GATES IN VARIOUS TECHNOLOGIES OTHER THAN QUANTUM CIRCUIT TECHNOLOGY .....	56
Chapter 4: REVERSIBLE QUANTUM CASCADES AND THEIR SYNTHESIS .....	112
Chapter 5: SEARCH ALGORITHMS VERSUS SEQUENTIAL ALGORITHMS FOR BINARY REVERSIBLE CASCADE SYNTHESIS WITH NO ANCILLA BITS .....	134
Chapter 6: SYNTHESIS OF REVERSIBLE CASCADES FROM INCOMPLETE AND RELATIONAL SPECIFICATIONS .....	186
Chapter 7: CONCLUSIONS.....	236
References .....	243
Appendix A: Glossary.....	253

## List of Figures

Fig. 2.1. Generalized Feynman Gate a) general schematics. b) an example in ternary logic. c) the truth table of the circuit from Figure 2.1b. ....	29
Fig. 2.2. Generalized 3*3 Toffoli Gate size .....	31
Figure 2.3. Generalized $n \times n$ Toffoli Gate.....	32
Fig 2.4a. Generalized 4*4 binary Fredkin Gate. This schematic emphasizes that a Fredkin gate which is a controlled-SWAP uses multiplexers to execute the SWAP. Here $f_2$ is an arbitrary function of A and B used for control of the multiplexers.....	34
Fig 2.4b. The schematics of 4x4 double-controlled Fredkin gate from Fig. 2.4a to explain controlled-SWAP.....	34
Fig 2.4c. 4x4 double-controlled Fredkin gate realized using two Feynman gates and 4x4 Toffoli gate. This schematic emphasizes that Fredkin family of reversible conservative gates can be built using Toffoli family of reversible gates. ....	35
Fig. 2.4d. Standard 3x3 Fredkin gate as a controlled-SWAP. ....	35
Fig. 2.5. Generalized $n \times n$ Fredkin Gate .....	36
Figure 2.6. Generalized 4*4 Kerntopf Gate. ....	38
Fig. 3.2.1.: Realization of NOT in double-rail technology. Every horizontal signal at the right corresponds to two physical wires. ....	64
Fig. 3.2.2: Realization of Controlled-Not (Feynman) gate in double-rail logic. ....	64
Fig. 3.2.3: Realization of Fredkin gate using Toffoli and CNOT gates. By surrounding the Toffoli gate (Fig. 3.2.3a) with CNOT gates (Fig. 3.2.3b) we obtain the Fredkin gate (Fig. 3.2.3c). Similarly the Toffoli gate can be obtained by surrounding the Fredkin gate using two CNOT gates. This way a Toffoli gate in double-rail logic can be realized, as shown in Fig. 3.2.4.....	65
Fig. 3.2.4: Realization of Toffoli gate in double-rail logic using a double-rail Fredkin gate surrounded by two Controlled NOT gates.....	66
Fig. 3.2.5.: Detailed circuit of double-rail Fredkin gate realized using two standard Fredkin gates controlled by the same signal $a_2$ .....	67
Fig. 3.2.6: The gate from Figure 3.2.5 in another notation, every signal corresponds now to two wires from Figure 3.2.5. ....	67
Fig. 3.2.7: Detailed circuit of double-rail “new gate” realized with three standard Fredkin gates. ....	68
Fig. 3.2.8: Realization of CNOT in double-rail technology. Signal “a” is represented by $a_1=0, a_2 = 1$ . Signal $a'$ (negation of a) is represented by $a_1=1, a_2 = 0$ . ....	69
Fig. 3.2.9: Realization of SWAP in double-rail technology.....	69
Fig. 3.3.1: Realization of Fredkin gate. This gate uses the notation for De Vos double-rail technology from Figure 3.3.2. ....	72
Figure 3.3.2. Realization of function Q in Fredkin gate using De Vos technology. Similarly function R from Figure 3.3.1 is realized. ....	72
Fig. 3.3.3: Realization of Toffoli gate in double-rail DeVos technology. As we see, this is a kind of double-rail technology, with wires a and $a'$ representing signal a, and so on for signals b, c and P. ....	73
Figure 3.3.4. Realization of CNOT in De Vos technology.....	74
Fig. 3.3.5: DeVos CMOS circuit layout for pass-transistor diagram from Figure 3.3.3.....	75
Figure 3.4.1: Y-gate .....	83
Figure 3.4.2: inverted Y-gate .....	84
Fig. 3.4.3: Realization of CNOT in double-rail switch-based technology, using 2*3 switch gates. ....	81
Fig. 3.4.4: Realization of Fredkin gate in single-rail switch-based technology, using 2*3 switch gates.....	82
Fig. 3.4.5: Using Switch gates to build a new reversible conservative gate. Observe that in this circuit a loop of gates is allowed. ....	85
Fig. 3.4.6: Schematics of the new gate based on the schematics from Figure 3.4.5. This gate can realize products, inhibitions, sums and implication functions on its three outputs. ....	85

Fig. 3.4.7: Fredkin gate with ordering of gates created by the algorithm.....	86
Figure 3.4.8. Schematic of a new single-rail reversible gate using Fredkin gates built from 2*3 switches.87	
Figure 3.4.9. Another way of drawing the circuit from Figure 3.4.8 to emphasize its similarity to generalized gates introduced in Chapter 2. Both muxes are controlled by the Exor. This circuit, in contrast to circuits in other figures is drawn from right to left rather than from left to right to emphasize its reversible nature and to help using the analysis from Figure 3.4.10.....	87
Fig. 3.4.10: Another notation for the circuit from two Fredkin gates given in Figure 3.4.8. Assume $x=c$ , $y=d$ .....	88
Fig. 3.4.11: Analysis of circuit from Fig. 3.4.10. ....	88
Figure 3.5.1: Y-gate generation tree .....	99
Figure 3.5.2: Single-rail Fredkin gate implementation.....	106
Figure 3.5.3: Double-rail Fredkin gate implementation. ....	108
Figure 3.5.4: Double-rail Feynman gate implementation.....	109
Figure 3.5.5: Double-rail Toffoli gate implementation .....	110
Figure 4.1.1. General cascade composed, from left to right, of Kerntopf, Toffoli and Fredkin gates.....	118
Figure 4.1.2a. Example of a multi-output Fixed Polarity Reed-Muller cascade from Toffoli Family of gates. (a) Reversible notation, (b) quantum notation has a more realistic gate model. Observe that each input variable has the same polarity in entire circuit. ....	119
Figure. 4.1.2b. Yet another notation that shows more details of gates used. The quantum costs [Lee07] can be calculated for each gate from this cascade. This cascade is called the quantum array in many papers. Observe the role of the mirror gate that reintroduces constant value of the ancilla bit. This is used in quantum oracles. ....	120
Figure 4.1.3. Example of a multi-output Fixed Polarity Reed-Muller cascade from Toffoli Family of gates. This is the same circuit as one from Figure 4.1.2 a, b but in yet another notation. ....	121
Figure 4.1.5. CMOS notation example of a multi-output ESOP (mixed polarity) from Toffoli Family of gates. Observe that variable A has different polarities in various Toffoli gates of the circuit.....	122
Figure 4.1.6. Notation used in quantum circuits which emphasizes the repetition of gates. ....	123
Figure 4.1.7. “CMOS circuit” notation for full adder realized using composition. Here we use To for 3*3 Toffoli, and Fe for Feynman gates. ....	124
Figure 4.1.8. Example of a multi-output circuit – an adder. This is the circuit from Figure 4.1.7 in quantum array notation.....	124
Figure 4.1.9. An adder with gates on only neighbor wires. Such gates are used in Ion Trap Quantum Realization technology. ....	125
Figure 4.1.10. The adder from Figure 4.1.7 with vertical lines showing levels of reversible gates. ....	126
Figure 4.1.11. Adder from Figure 4.1.10 with intermediate signals illustrating steps of its analysis. Example of application of a level-by-level synthesis algorithm applied to our example. ....	126
Figure 4.1.12. Quantum array notation for Full Adder realized using composition. The width of the “quantum register” is four. Note another order of inputs. ....	128
Figure 4.1.13. Quantum array notation for Full Adder realized using PPRM type of logic. This design style is used in quantum oracles in which all inputs must be repeated to outputs.....	128
Figure 4.1.14. Synthesis of large Toffoli gate from 3*3 Toffoli gates and next from 2*2 quantum primitives such as Controlled-V and Controlled-V+. These primitives are not explained in the dissertation. The only goal of this figure is to show the approximate method to calculate quantum costs. The quantum cost is approximately the number of 2*2 quantum primitive gates. This figure shows also the use of ancilla bit. (a) a 4*4 Toffoli gate, (b) decomposition of a 4*4 Toffoli gate to three 3*3 Toffoli gates that uses one ancilla bit initialized to zero, (c) further decomposition of circuit from b to 2*2 quantum primitives.....	128
Fig. 4.2.1: The use of mirror circuit $G^{-1}$ to an arbitrary reversible circuit G in a general-purpose reversible circuit with 2 outputs. ....	130
Figure 5.2.1. MMD method illustrated with truth tables of intermediate functions. Notation $a \rightarrow c$ means $c$ $= c \oplus a$ means “flip c if a=1”. Control lines are <u>underlined</u> and affected bits are shaded. ....	140

Figure 5.2.2. The solution circuit found from MMD in Fig. 5.2.1 drawn and created from outputs to inputs. The arrow shows the flow of signal from inputs to outputs. This method is possible because each reversible gate used in this figure is its own self-inverse. ....	142
Figure 5.2.3. (a) Graphical visualization of MMD algorithm. A Branch of a search tree is shown with the final function as the initial node (K-Map). Arrows correspond to applying operators which in MMD are Toffoli and Feynman gates (only Toffoli here). (b) The final solution, non-minimal. The intermediate signals $A^1, B^1, C^1$ are created after applying Toffoli gate to primary outputs A, B, C. Similarly after applying the second Toffoli gate, the intermediate signals $A^2, B^2, C^2$ are created. The next signals created are $A^3 = a, B^3 = b, C^3 = c$ , so the search is terminated as identity has been found. The first map in (a) shows the situation when we flip bit c in these cells in which bits ab = 11. The second map is when we flip bit b in cells in which a = 1 and c = 1 (these bits are shown bold). The third map shows the synthesis stage when we flip bit c in these cells in which bits a = 1, b = 1. The final map at the bottom is when reaching identity completes the search. Every branch of our search tree terminates with a leaf that is either an identity or a branch is interrupted because the search in this branch is evaluated as not prospective and not leading to a potentially better solution. ....	143
Fig. 5.2.4. Another search for function from example 5.4.2.1. (a) Function A(a,b,c), (b) Function B(a,b,c), (c) Function C(a,b,c), (d) Search tree with two branches. The upper branch is not completed. The lower branch leads to the improved solution from Figure 5.2.5. ....	145
Figure 5.2.5. The optimal circuit found by modifying the search in MMD. It is created in the lower branch of the tree from Figure 5.2.4, gates created from outputs to inputs. ....	146
Figure 5.3.1: Design of Fredkin gate. (a) Optimal order shown in the branch of the search tree above violates the natural order used in MMD (MMD order), (b) the solution circuit found in the process from Fig. 5.3.1a which is the well-known minimal circuit of Fredkin gate. ....	151
Figure 5.4.1. New orders for MMD-like synthesis. (a) Hasse diagram with binary vectors, (b) Hasse diagram with natural numbers, (c) Ordering of nodes that violates the MMD order, illustrated on the Hasse Diagram. This is however a valid MMDS ordering. ....	154
Figure 5.4.2. New ordering 02134657 for MMD-like binary synthesis, a valid MMDS order which is consistent with the Hasse diagram relations of orderings. ....	156
Figure 5.4.3. Graphical visualization of MMDS orderings in Karnaugh Maps and explanation of the concept of blocking, also the rule and enumeration of cells of the KMap. ....	158
Figure 5.4.4. Graphical visualization of MMDS orderings in Karnaugh Maps. Beginnings of the orderings. ....	159
Figure 5.4.5. Graphical visualization of MMDS orderings in Karnaugh Maps. Orderings visualized with colors. ....	160
Figure 5.4.7. Graphical visualization of MMDS orderings in Karnaugh Maps. MMD order was added to MMDSN to have results always not worse than MMD CAD tool. ....	162
Figure 5.5.5. The visualization of the MP algorithm. Fourth stage. ....	169
Figure 5.5.6. The visualization of the MP algorithm. The final stage. ....	170
Table 5.2. ....	185
Figure 6.1.1. Incomplete Permutation Matrix for example 6.1.1. ....	189
Figure 6.1.2. General Decomposition of Unitary Matrix U into a sequential composition of blocks X and Y. Relational specification U is decomposed to a composition of gate Y that comes from our library of gates LIB and the remainder specification X. This procedure is repeated by decomposing X in the same way iteratively, until the remainder $X_n$ is found that has a matrix that can be completed to an identity matrix. This completes the synthesis. ....	190
Figure 6.1.3. Realization of gate Y from library of gates LIB. A Feynman Gate with Exor Up was assumed here as gate Y. The library stores every gate in the form of its permutative matrix, name and schematics. ....	191
Figure 6.1.4. Permutative Matrix $Y = Y^{-1}$ Please note that the matrix from the library LIB and its inverse matrix are the same. This is the property of the gates that we use. ....	192
Figure 6.1.5. Synthesis based on matrix multiplication starting from incomplete specification U for example 6.1.1. The algorithm found the relational specification of the (new) remainder function X. ....	193

Figure 6.1.6. (a) The Unitary matrix of gate $X'$ found by the algorithm, (b) the circuit corresponding to the unitary matrix $X$ , (c) the complete circuit synthesized for the original specification $U = YX$ . The circuit from Figure 6.1.6c corresponds to the general decomposition scheme from Figure 6.1.2.....	196
Figure 6.1.7. Permutative matrix $X''$ for the symbolic (relational) matrix $X$ .....	197
Figure 6.1.8. KMap for PQ, first together, then separated to KMaps for P and for Q.....	197
Figure 6.1.9. Second solution circuit obtained from the symbolic matrix in Figure 6.1.5. ....	197
Figure 6.2.1. Main Idea of MPS algorithm to synthesize incompletely specified functions as a reversible cascade with (potentially) ancilla bits. The first step converts function to backtrack completeable function. Such function is realized with ESOP minimizer for small functions. For larger functions the design is converted to completely specified function or is realized gate by gate with partial conversions of don't cares to cares.....	203
Figure 6.2.1.1. Synthesis of an incompletely specified 3*3 function as a reversible cascade using preprocessing algorithm. Separating the outputs of the KMap to individual outputs A(a,b,c), B(a,b,c) and C(a,b,c), and using the property of balancedness for each of the outputs to replace don't cares with cares. (a) The initial specification of the problem with don't cares, (b) Separating the outputs of the KMap to individual outputs A(a,b,c), don't cares on left for minterms 100 and 111 are finally replaced with 1's shown at right in cells 100 and 111, (c) Separating output B(a,b,c), (d),(e) Separating C(a,b,c) and using the property of balancedness for each of the outputs to replace don't cares with cares. ....	204
Figure 6.2.1.2. Karnaugh Map for signals A and B analyzed together to verify the reversibility. $A = ab' \oplus c$ . $B = a$ . The map shows pair of binary output vectors to be separated. A new output C must be added in such a way that each output vector A,B,C will be different. ....	205
Figure 6.2.1.3. Example for synthesis with don't cares using the MI-MO-IR algorithm. Illustrates realization of function C(a,b,c). From separation pairs in Figure 6.2.1.2, the cell 010 should be 0 and the cell 101 should be 1. ....	207
Figure 6.2.1.4. Reversible Cascade with one ancilla bit and one garbage bit synthesized for specification with don't cares and using the MI-MO-IR algorithm in which ESOP-based synthesis is internally executed. Output function $B = a$ was realized as the first output bit. Output function $A = a'b' \oplus c$ was realized as the second. Creation of this function required creating of garbage bit $\text{Garbage} = b$ . Function $C = a'b'c' \oplus ab' \oplus a'bc$ was minimized as the third output function. Observe that the created earlier output A was reused to minimize function C. This is one of main advantages of this method.....	207
Fig. 6.2.1.5. Synthesis with don't cares using the MI-MO-IR algorithm in which ESOP-based minimization is internally used. $C = A \oplus (a'b' \oplus ac)$ .....	208
Fig. 6.2.1.6. Synthesis with don't cares using the MI-MO-IR algorithm in which ESOP-based minimization is internally used. In this variant, in contrast to the variant discussed above where there was no order of outputs, we assume order of outputs B,C,A,G (G is a garbage). This leads to the necessity of calculating new truth table of function A, as shown in Figure a) and b). The function B cannot be completed without copying variable c, so one ancilla bit initialized to zero is added and used to create input for Toffoli gate in C. This way, the garbage $G = c$ , which is a common situation that inputs are copied to make function reversible. ....	210
Figure 6.2.1.7. Truth table for function from example 6.2.1 with assumed ordering of output variables BCAG, where G is a garbage bit. ....	211
Figure. 6.2.2.1. Synthesis of Fredkin Gate using "Multi-parameter Search Algorithm MPS" using tree search. Solutions found by "multi-parameter algorithm" are given in Figure 6.2.2.3. (left branch of the tree) and Figure 6.2.2.2. (right branch). Notation $ab \rightarrow c$ means executing $c = c \oplus ab$ gate. Cells left empty in left branch are self-mapping minterms. This notation emphasizes concentration on greedy maximizing the number of self-mapping minterms. The minterms currently processed are listed. The goals are to fix variables c and next b in order to minimize the number of bad wires. ....	213
Figure 6.2.2.2 The solution found by "Multi-parameter Algorithm". This is the circuit found in the right branch of the tree from previous Figure 6.2.2.1 .....	214

Figure 6.2.2.3 Non-minimal synthesis of the Fredkin Gate. This is the solution found by the MMD algorithm. It has a higher quantum cost than the solution found by the “Multi-parameter Search Algorithm”. This solution is also found in the left branch of the tree from Figure 6.2.2.1. Simulation for input data 101 is shown. This way we can verify results of our algorithms for all input vectors. .	215
Figure 6.2.2.5. The solution found by “Multi-parameter Algorithm MPS”. This is the circuit found in the right branch of the tree from previous Figure 6.2.2.4.(this is same circuit as in Fig 6.2.2.2).....	217
Figure 6.2.2.6. Detailed analysis of operations performed in the right branch of the tree from Figure 6.2.2.4. By symbol com near blue arrows we denote completion of don't cares to cares. By symbol com near read arrow we denote the final backtrack-completeable function obtained as a sequence of don't care completion. This function cannot be further completed to self-mapping minterms so a gate must be selected. The algorithm further converts don't cares to cares – this time with the goal of allowing application of gate $b = b \oplus c$ (denoted by $c \rightarrow b$ ). This completion is done to realize minterm 101. However, application of the gate destroys now the previously determined self-mapping minterms, for instance 001. This means that in future the operation should be undone with a mirror. This in reality happens with next operation of $c \rightarrow b$ operation in the previous to last stage. In the last stage the final don't care is converted to a care in the only possible way to keep the function to be completely reversible. As the result the algorithm obtains a completely specified identity function, the same as all our algorithms. ....	219
Figure 6.2.2.7. Truth Table of the initial specification for the incompletely specified 3*3 function .....	221
Figure 6.2.2.8. Output vectors generated by MPSRS in steps S1-S5. ....	222
Figure 6.2.2.9. Output vectors generated by MPS in steps S6-S10. ....	223
Figure 6.2.2.10. Output vectors generated by MPS in steps S11 - S15. ....	223
Figure 6.2.2.11. Output vectors generated by MPS in steps S16 – S20 with backtracking.....	225
Figure 6.2.2.12. Output vectors generated by MPS in steps S21 – S24. ....	225
Figure 6.2.2.13. Output vectors generated by MPS in steps S25-S29 with backtracking. ....	226
Figure 6.2.2.14. Output vectors generated by MPS in steps S30 – S34. ....	227
Figure 6.2.2.15. Output vectors generated by MPS in steps S35 - S38. ....	227
Figure 6.2.2.16. Output Original function versus final function $F(x, y, z)$ after MPS application. ....	228
Figure 6.2.2.17. The KMaps for P, Q and R separately obtained from the original incomplete specification in Figure 6.2.2.10. ....	230
Figure 6.2.2.18. The truth table after applying the first gate from inputs, with output in wire P'P.....	230
Figure 6.2.2.19. Realization of output functions Q and R as functions of variables P' y and z. $Q = y'$ , $R = y' \oplus z$ .....	231
Figure 6.2.2.20 Cascade realized by the second variant of synthesis for Example 6.2.2.3. This method uses the algorithm from section 6.2.1.....	232
Figure 6.2.2.21 The final truth table after applying all gates.....	232

## List of Tables

TABLE 1.2.1 Goals of this dissertation.....	6
Table 3.4.1. The truth table for Fredkin gate to be used in next examples.....	79
Table 3.4.2: Y-Gate truth table .....	8080
Table 5.3 Algorithm Comparison Table .....	183
TABLE 6.2.2.1. Comparison of results of MPS and DCARL on quantum costs.Orig means DCARL, new means MPS. Names of functions have percent of don't cares as the second part of the name. The first part of the name tells how many qubits the function has. ....	233
Table 7.1 Goals of the dissertation and theories developed.....	238

## Chapter 1: INTRODUCTION

### 1.1. Introduction

This dissertation is devoted to efficient automated logic synthesis of reversible circuits. These Reversible circuits are of interest to several modern technologies, including Nanotechnology, Quantum computing, Quantum Dot Cellular Automata, Optical computing and low power adiabatic CMOS, but so far the most important practical application of reversible circuits is in quantum computing. Logic synthesis methodologies for reversible circuits are very different than those for classical CMOS or other technologies. My interest in this dissertation is only in logic synthesis aspects and not in physical (technological) design aspects of these circuits. This research is not related to quantum mechanics. I will use only some elementary concepts of quantum mechanics and notation, as they will be useful to explain the concepts of quantum costs.

#### Comments and explanations.

1. Quantum circuits and gates are described by unitary matrices. 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.

2. 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, 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. In this dissertation, I am interested in the permutative (reversible) subclass of quantum circuits as they play a very important role in many quantum algorithms. For example, the oracles of the famous Grover Algorithm are quantum permutative, so mathematically they are reversible circuits as designated in this dissertation.
3. The main problem of synthesis of quantum circuits (discrete in contrast to analog or continuous) is to start from a unitary matrix  $\mathbf{u}$  specification ( $\mathbf{u} \times \mathbf{u}^* = \mathbf{I}$ ) of a circuit and decompose this initial specification to unitary matrices of realizable “quantum gates” such as Hadamard gates, Feynman or Toffoli gates. In this dissertation, I am solving a subset of this problem by assuming that the unitary matrix is permutative. Thus, the corresponding circuit can only include gates such as NOT, Feynman, and Toffoli.
4. 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  $\log 2 \times KT$  where  $K$  is Boltzmann’s constant and  $T$  is the temperature. Reducing power became the main task of modern digital

circuit design, making design with reversible circuits of more interest as it reduces power that is dissipated by computing systems. Reversible circuits have thus applications also in new technologies outside the scope of quantum computing.

The focus of this dissertation is on synthesis of reversible (permutative) binary circuits. It is not related to general unitary circuits that are used in quantum computing and which exhibit quantum mechanical phenomena such as superposition and entanglement.

You may ask, why the permutative reversible circuits are so important and should be a topic of a dissertation, rather than the general quantum circuits specified by arbitrary unitary matrices? The answer is that the permutative quantum circuits include the class of oracles and blocks that are parts of oracles, such as comparators or arithmetic blocks, counters of ones, etc. Every practical quantum algorithm, such as the Grover Algorithm, has many permutative circuits. These circuits are also used in Shor Algorithm (integer factorization), simulation of quantum systems, communication and many other quantum algorithms. Designing permutative circuits is the major engineering task that must be solved to practically realize a quantum algorithm.

In addition to quantum circuits that are reversible, it is expected by many researchers that reversible circuits will be built for many new nano-technologies other than quantum. There is very little published on the logic design aspects for these non-quantum reversible technologies. For example, there is no published paper on designing using conservative logic.

## **1.2 Goals of This dissertation:**

### **1.2.1 What this dissertation is about: Synthesis of reversible circuits.**

Reversible computing is of high importance to future computing, and ultimately, every future computing technology will include reversible circuits. Therefore as the background of this dissertation I will superficially introduce several technologies, including quantum technology, in which circuits are reversible. I will not go to the physical principles of designing such circuits. Next I will present much improved and new methods for synthesizing these circuits, also those that start from new types of specifications. These methods will be superior to previous methods in the sense given in table 1.2.1 below.

### **1.2.2 What this dissertation is NOT about?**

The physics of quantum gates and their quantum mechanical properties such as superposition and entanglement are not central to this dissertation. Our interest is only in classical behaviors of quantum circuits, which means, quantum permutative circuits behaving like reversible circuits. We are also not interested in the technological aspects of quantum gate realizations. Similarly, as a dissertation in classical logic synthesis may not go into electronic construction of a NAND gate, in this dissertation I am not

interested what is inside a quantum gate. I just assume, based on the literature, few selected types of reversible quantum and non-quantum gates. I extend and generalize these gates independent on their realization technology. Next I use them in my logic synthesis procedures. In classical logic synthesis, the combinational logic synthesis is a separate research subject from Finite State Machine (FSM) design, but designing a combinational circuit of excitation and output logics is used as a part of the so-called structural synthesis of FSMs. The same is true for quantum circuits, the reversible (permutative) combinational circuits are parts of quantum systems with memory and quantum automata. These topics are not described in this dissertation, but the knowledge of the fact that combinational quantum circuits are used to build sequential quantum circuits highlights the importance of developing systematic synthesis methods for them.

I will concentrate only on the logical structures of reversible gates and circuits. I will also consider the “realization costs” of circuits in these technologies (i.e., costs of circuit realization). These costs are related to circuit speed, power loss and circuit complexity. As neither myself nor my Advisor are physicists or technologists, I will base my selection of reversible technologies on the opinions of top world authorities in this area and the technology-related papers that can be found. Even the physics and technology experts differ deeply as to the relative importance of various reversible technologies. Therefore, I will briefly review several types of reversible technologies (but not all variants of each technology). Similarly, when possible, I will take the basic costs of quantum (reversible) gates in various technologies from papers published in the literature. The costs of gates

for all technologies will not be provided herein, because such enumeration is not necessary to show the efficacy of the new algorithms I proposed. The costs of quantum reversible gates will be taken from technology-related publications of other authors, for example, Soonchil Lee et al [Lee06] and Maslov [Maslov03].

The goals of this dissertation are specified in table 1.2.1 below.

*TABLE 1.2.1 Goals of this dissertation*

<b><i>Goal number</i></b>	<b><i>Goal formulation</i></b>	<b><i>Competitor to be measured against</i></b>
Goal 1	Being able to synthesize reversible arrays (circuits) for completely specified reversible functions of such large size that there exists no tool to synthesize them. The size of the function is expressed in terms of number of variables, number of Generalized Toffoli gates and cost (such as quantum cost).	MMD software of Miller, Maslov and Dueck, software of Agrawal and Jha.
Goal 2	Being able to synthesize reversible circuits for functions with “don’t cares” in their initial specifications. The cost of the synthesized circuits should be smaller	DCARL of Manjith Kumar

	than cost obtained using the only available tool DCARL	
Goal 3	Being able to synthesize reversible functions with circuit cost related to several technologies such as conservative, adiabatic CMOS, quantum or optical technologies, and not just the number of gates or literal as cost.	No theory or CAD tool exists that can do this.
Goal 4	Being able to synthesize reversible circuits for new types of reversible logic, for which no algorithm was ever created. (Y-switches)	No method, theory or CAD tool exists that can do this.
Goal 5	Being able to synthesize a reversible circuit from a functional specification that is not reversible.	DCARL can do this in a limited way.
Goal 6	Being able to synthesize reversible circuits which have smaller costs than costs produced by the top tool, MMD. Costs will be defined differently for various technologies.	MMD, software of Agrawal and Jha, my previous attempts and variants of software developed by me.

Below I will explain in full detail many concepts and terms that I used in Table 1.2.1 above.

In coming chapters of my dissertation, I will relate to these goals and show how these goals were satisfied by theories and software developed in this dissertation. Note that improved processing speed is not a goal of my software. The ability to synthesize large functions and reducing cost of synthesized circuits are the most important properties of CAD approaches as this is a common opinion of researchers in this area. It is also very important to understand current limitations for synthesis of large reversible circuits from the algorithmic point of view.

### **1.3. Universality versus specialization of the methods I proposed**

In this dissertation we will integrate and compare previous approaches for synthesis of reversible circuits. Although our general synthesis algorithms apply to all reversible logic technologies (such as CMOS and optical, in particular), our evaluation methods are geared towards selected quantum technologies such as Nuclear Magnetic Resonance (NMR) and ion trap. All algorithms proposed in this dissertation can be tuned towards certain technologies by simply incorporating that technology's cost functions into the synthesis process for these algorithms.

#### **1.4 Why do we need CAD methods for synthesis of reversible and quantum circuits?**

Automatic synthesis of reversible circuits is an important component of the entire reversible computing technology, simply because future CAD systems for reversible computers will use such design software. This task is important by analogy as we all understand the importance of logic synthesis CAD for classical binary computers. The methods developed in this dissertation can find applications in “by-hand” synthesis, and next in automated synthesis of reversible technologies such as quantum, optical, nano, DNA and CMOS circuits. The most mature of these technologies is the quantum technology, so I concentrate on gates that are used by other authors in quantum circuits. As quantum computers are becoming larger (with more qubits) every year (since 2001) and will perhaps be commercialized in coming years, there is a need to develop efficient Computer Aided Design (CAD) methods, even for these quantum computers that already exist. For example, in November of 2007, Canadian Company DWAVE have demonstrated a 28-qubit adiabatic quantum computer this is an “analog” type of quantum computer which is different than the discrete “quantum circuit” model of quantum computer that is of interest to this dissertation. However, the two models are equivalent and every circuit that we develop here could be converted to the quantum adiabatic model. Several research groups have quantum computers (classical quantum circuit model discussed here) with around 10 qubits.

Automatically synthesizing large quantum circuits is one of the most important prerequisites to build a practical quantum computer [Nielsen00]. In the case of truly quantum circuits, the practice of quantum circuit design was already ahead of theory in year 2004, as only circuits with 3 quantum bit (qubit) could be designed automatically (exact minimum), while there already existed a 7-qubit quantum computer of the “Circuit Model” type. There is currently no method in the world to design an exact optimum reversible circuit for arbitrary permutative function of 4 bits. (The bi-directional synthesis algorithm based on group theory [Yang05] is currently the software for exact synthesis (guarantees minimum cost solutions) that can synthesize the highest percent of 4-qubit circuits.) The state of the art in 2010 is synthesizing quantum circuits for approximate minimum costs, with not more than 12 qubits. This dissertation improves on previous research as our method allows finding optimal or high quality results (in terms of cost) for all 4-qubit circuit synthesis [Yang05]. Finding the library of least-cost solutions for each 4-qubit binary reversible function will be useful in the future because of the existence of new hierarchical decomposition methods based on quantum multiplexers [Khan06]. Most importantly, we can synthesize circuits with 30 qubits. In other areas of reversible circuits design, the design practice is also ahead of the level of the synthesis tools, quantum dot reversible technology is an example. In other nano-areas, the synthesized gates (circuits) are very small (optical, nano), that my methods can synthesize circuits of practical size right away. The MP algorithm I created, has already synthesized the largest quantum circuits ever synthesized by software.

### **1.5 Previous work on reversible logic synthesis.**

In the first part of my dissertation, I present and analyze important published papers about synthesis of binary reversible circuits. Many such papers have been published world-wide in the last 20 years (most of them were published in the last 8 years). The field of synthesis of binary reversible (quantum) logic started very recently (after year 2000). I have searched data bases such as Google Scholar, to find relevant papers, I have also (with the help of my advisor), familiarized myself with all algorithms and software that exist for synthesis of permutative circuits. And studied different algorithms and search heuristics that are useful in my research. The research presented in this dissertation combines ideas from various areas of research. My research is the second that applies AND/EXOR FPRM (Fixed Polarity Reed-Muller) minimization to reversible logic, and the first that applies ESOP (Exclusive-Or Sum of Products) “logic minimizer” for reversible cascades besides wave cascades of Mishchenko and Perkowski [Mishchenko02].

### **1.6 Library based design.**

In classical logic synthesis, there is an area of “library based design”, in which the designer does not have to design the logic of his circuit on the level of basic primitives such as transistors. Instead, he synthesizes with “ready cells” (library cells) such as AND-

OR-INVERT or EXOR. Similarly, the optimization software in the CAD tools work on the level of characterized gates (cells) from the library. This dissertation is not a technology or circuit related dissertation, but rather a logic synthesis dissertation. Thus, I follow here the approach of library based design. Based on literature, I extract the gates realized in various technologies and make a library of these gates. Our design methods for all kinds of reversible technologies can be characterized as “library-based” synthesis as gates exist with their characterization in a library of gates and the software calculates the final total cost of the synthesized circuit as a sum of costs of cells used to build this circuit. These costs can be of various meanings for different technologies, though.

### **1.7 Innovative aspects of this dissertation.**

The next goal of this dissertation is to show areas of reversible circuit design for which no research has been done at all, and that are important to future design of reversible circuits, for example, synthesizing reversible automata. By analogy with classical logic design, there is a need to develop such aspects of design automation as, for example, synthesis from incomplete specifications (don't cares and relational). Incomplete functions always appear when synthesizing automata and they also appear in synthesis of quantum automata [Kumar10]. I will also present results obtained in other research areas of reversible logic synthesis and present improvements to these results by the new ideas and algorithms that I developed in this dissertation.

## **1.8 Uniform presentation of methods.**

I will also discuss in a uniform way several gates and circuits from various reversible technologies and their variants that are already proposed in the domain of reversible computing. These gates exist in many papers written by physicists, logicians, mathematicians and engineers. In this dissertation, we will present reversible gates from the most important and known “reversible technologies” discovered in years 1980 to 2010. I will strive to unify the approach to gates that are built in these technologies; their possible generalizations and extensions. Next my dissertation will completely abstract from the reversible technologies, and concentrate only on mathematical, logical and algorithmic aspects of reversible circuit design. These are the core topics of my dissertation.

In the second part of my dissertation, I will present my original unified methods to synthesize reversible binary circuits from specification methods used in classical synthesis. My dissertation is the first piece of research that uses these specifications in reversible synthesis.

## **1.9 Synthesis of incomplete functions and relations.**

My synthesis methods will be extended to specifications of circuits that are incomplete (with don’t cares) and relational (describing Boolean relations, constraints on values that

are more complex than don't cares). This is a new research area. There is only one M.S. thesis from PSU on synthesis from incomplete specifications. There are entirely no papers on synthesis from relational functional specifications for quantum and reversible circuits.

### **1.10. My software.**

In this dissertation I will discuss the experimental results of my software based on some theories and algorithms outlined above. As the technology is ahead of CAD in the area of reversible logic synthesis, it is very important to develop new software that can be used to synthesize large functions and that will also have additional functionalities such as the option of synthesizing without adding ancilla bits. Such software was created during my research. This software extends and improves software for reversible logic created by various research groups including PSU. My software exceeds all results published in the literature. This software is able to solve new problems that remained unsolved as of year 2010 and achieves goals from Table 1.2.1 in section 1.2. I slowly created and improved my software over many years. Although the experimental results of my initial MMDS software were good in terms of circuit costs, the method was limited to small functions. In last year and a half, already after the “dissertation proposal” meeting with the Committee, a breakthrough occurred when I found new realization of functions to be used with this algorithm. This allowed applying my modified software MP to large functions. My goal of developing the improved software was not to play the “speed

competition”, but was rather a necessity for making reversible CAD software useful for large practical circuits, for example, in quantum circuit design of NMR computers [Lee06]. The good method should be scaleable (in terms of size of function being synthesized) and should allow for various types of function specifications to be used. Table 1.2.1. illustrates goals related to these ideas.

### **1.11 Motivation for Reversible Logic**

The only goal of using Reversible Logic in technologies other than quantum is Low Power Design. One may ask, what are the limits of low-power design? IBM Corporation was the first company that asked this question in the 1960’s. The answer was given in the famous paper of Bennett and Landauer [Bennett73]. These researchers proved that losing information in a circuit is equivalent to losing power. *“Whenever we lose information some power is also lost”*. Thus 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. Information is lost when it is transmitted through the AND gate regardless of the technology in which this gate is realized. Every classical logic gate, other than inverter, has this unfortunate property.

If the performance of computational systems is to be improved further, it is essential that the energy dissipated by each logic operation is reduced. This can be achieved, to certain extent only, by implementing the logic circuit in a reversible technology, the technology in which information is not lost. Information is not lost during computing, it is lost only when reading and writing. This is a very important theoretical result that gives direction to all future technologies.

The importance of this information-theoretical aspect of power reduction will only increase with the progress in technology in 21<sup>st</sup> Century. As part of the energy lost in a gate is related to the technology and another part of energy lost in this gate is due to the information loss, we will pay attention in this dissertation only to the second part which so far, in year 2010, is much smaller than the technology-related component, in all existing technologies. But this part will become dominant with ongoing progress in technology-related low power design and the arrival of new technologies such as quantum dots. We can thus say that my research interest in this dissertation is in the technology-unrelated, or in the information-theory-related, aspect of low power design. This aspect is more mathematically oriented than physically or technologically oriented. The energy-related results of Landauer, Bennett and Benioff were proven and confirmed by many other authors. Again, it is not a goal of this dissertation to go deeply to physics and thermodynamics power aspects explained in these works, but only to use the results from these areas as a motivating factor to design reversible circuits, both quantum reversible circuits and non-quantum reversible circuits.

The gate that does not lose information is called “reversible gate”. For example, the 2-input 2-output Feynman (CNOT) gate (we will call it a 2\*2 gate, for short) described by the equations  $\{P = A, Q = A \oplus B\}$  is reversible, because for each combination of the output signals  $\{P, Q\}$ , there is exactly one combination of the input signals  $\{A, B\}$  that corresponds to it. Logic circuits consume energy because of technological factors (such as leakage or power dissipation while switching) and also as a result of losing information. This second component (loss of information), was made important because of a series of papers by Landauer, Bennett, Benioff and other researchers in the last 30 years [Landauer61, Bennett73, Bennett82, Bennett89]. The first component is constantly decreasing due to the improvement of the implementation technologies and the emergence of new design principles such as the adiabatic design. The adiabatic design principles are used in CMOS, quantum dot, fluidic and truly quantum technologies. It becomes the main idea of future computing. The second component of the energy consumption is related to information and can be decreased (to zero) only by adopting reversible design principles. As of year 2010, the second component of energy consumption is still much smaller but if the progress in low-power technology follows Moore’s law, the second component will start dominating around years 2020-2030. According to Landauer [Landauer61], this second component is a necessary condition for future design. It suggests using only reversible gates when building a logic circuit that does not waste energy. Using only reversible gates in design is however not a sufficient condition of reducing power.

It was experimentally shown that reversible adiabatic gates can be built in CMOS [Athas], [Younis94], DNA, several nano, optical [Cuykendall87, Gilchrist03] and other technologies, and that quantum logic gates in all technologies are naturally reversible (because these gates are mathematically described by unitary matrices, and that unitary matrices are reversible, the physics of all quantum circuits dictates that all these gates are reversible) [Nielsen00, Muthukrishnan00]. The measurement operation in quantum computing, is however not reversible. In this dissertation, we are not interested in the important problems of quantum circuit initialization and measurements, which is the phenomena responsible for the interesting peculiarities and powerful computational abilities of quantum computing. Our interest is only in reversible logic synthesis using reversible gates, which are important parts of quantum circuits.

Based on facts mentioned above in regards to future technologies and opinion of authorities in this area, although we cannot predict what new technologies will be proposed in the future, and which technologies will win the competition for the successor of CMOS as a commercial technology, we can state with a very high degree of certainty that ultimately all these forthcoming technologies will use some sort of “reversible circuits”. These opinions are expressed by top experts from high-tech companies and US Government institutions (reports from IBM and Sandia [DeBenedici07]). So positive results from this dissertation (creating efficient

synthesis algorithms for large reversible circuits), will have long-lasting and technology-independent importance for future computing systems.

## **Chapter 2: BACKGROUND AND LITERATURE REVIEW ON BASIC REVERSIBLE GATES**

### **2.1. Background on Reversible Logic and Reversible Logic Synthesis**

Although our main interest in this chapter is in designing permutative (quantum) circuits, our methods have also relevance to designing classical reversible circuits that can be realized in CMOS, nano-technology or optical technologies. The gate that does not lose information is called *reversible*. 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). Because of these non-typical requirements, reversible gates are quite different than classical gates and synthesis methods are different as well.

It was shown that reversible gates can be built in DNA, Quantum Cellular Automata (QCA), Quantum Dot, optical, nano and other technologies, but most research efforts on reversible circuits realizations (other than quantum) have been devoted to CMOS circuits [Athas, Younis 1995]. The synthesis methods based on decision diagrams such as Binary Decision Diagrams BDDs, Kronecker Functional Decision Diagrams KFDDs and Pseudo-Kronecker Functional Decision Diagrams PKFDDs

have been adapted to reversible logic [Perkowski97f]. Some methods are based on the composition of gates to circuits, other methods decompose general specifications to elementary specifications (gates). These methods have been created as adaptations of classical synthesis methods. Many of these approaches suffer however from requiring a very high numbers of ancilla bits to be added during synthesis. Adding only few ancilla bits may however be very useful in synthesis, therefore, this dissertation will concentrate first on synthesis methods that add no ancilla bits, and then on methods that add very few ancilla bits, possibly one, to show that synthesis can be improved with respect to the number of gates at the cost of having only a small increase of the number of bits (the so-called width of the “quantum register”, in another terminology).

Most of reversible gates from literature are three-input three-output ( $3 \times 3$ ) or four-input four-output ( $4 \times 4$ ) gates. The exceptions are [DeVos00, DeVos01] in which restricted multi-input, multi-output gates were presented without any general systematic synthesis methods. In this dissertation, we will introduce  $n$ -input  $n$ -output reversible gates for  $n > 4$ . To our knowledge no systematic methods for synthesis using arbitrary gates (other than Toffoli and Fredkin) with  $n > 3$  was ever published with the exception of Miller et al and Agrawal et al [AgrawalJha04b, Jha06], which were however restricted to Toffoli/Fredkin families of gates.

Below we will present families of  $n$ -input  $n$ -output ( $n \times n$ ) gates for arbitrary value of  $n$ , as well as practical systematic synthesis methods from literature for small and medium values of  $n$ .

## 2.2. The essence of Reversible Logic Synthesis

To avoid energy loss, the approach originated by Ed Fredkin and Tommaso Toffoli [Fredkin82] (and most subsequent early authors) has been next commonly used. It creates a “basic circuit” from reversible gates with garbage outputs. Next, this approach applies a “spy gate” for every primary output. The spy gate is a Feynman gate with  $B = 0$  which copies the output signal of the basic circuit. Next a mirror circuit is added with inputs from second outputs of spy gates and garbage outputs of the basic circuit. The mirror circuit is the inverse of the basic circuit and has as many gates (that are inverses to the gates in the basic circuit) as there is in the basic circuit. This solution leads to the duplication of the circuit’s delay and cost of gates. The delay is  $2n+1$  where  $n$  is the delay of basic circuit, and the gate cost is  $(3m + k)$  where  $m$  is the number of gates and  $k$  is the number of primary outputs (and spy gates). Therefore, all methods that do not take garbage bits into account while designing the basic circuit lead to very inefficient results. The main design requirement is that reducing the garbage in the basic circuit (ideally reducing to zero

garbage, i.e. no garbage at all) is a good approximation for all quasi-optimal logic synthesis algorithms that use reversible gates. Remember that in the so-called “oracles” used in quantum algorithms (Grover) [Grover96], the inputs should be forwarded to outputs. These qubits are thus not treated as garbage bits. When designing quantum oracles, there should ultimately be no garbage qubits in a circuit realizing the Grover Loop. Mirror circuits are added to return all ancilla qubits to constant states, usually a 0. This condition is one of the constraints for synthesis methods developed in this dissertation.

### **2.3. Differences between Classical Logic and Reversible Logic**

It is very important to understand differences between non-reversible circuits and reversible circuits from the synthesis point of view. The differences of reversible logic synthesis compared to binary logic synthesis affect very deeply the mathematical and algorithmic aspects of synthesis. They can be summarized as follows:

1. While classical logic gates have many inputs and one output, the gates used in most reversible logic technologies (especially in quantum computing) have equal numbers of input and output signals. We use this model of gates in this dissertation, called  $k \times k$  gates.

2. However, in very few (but important) reversible (non-quantum) technologies, the condition of equal number of inputs and outputs is not satisfied. We will show relations between these other types of reversible gates and the  $k \times k$  type of gates. These  $n \times m$  gates can have more outputs than inputs or vice versa. These gates, whether proposed by other authors, or new gates that I proposed, will be discussed in chapter 3. I show that even if basic reversible gates have 2 inputs and 3 outputs in these technologies, they can be used to build standard reversible gates and circuits that have same number of inputs and outputs, therefore, they can be used by the synthesis methods that I developed in chapters 5 and 6. The requirement of having the same number of inputs and outputs in reversible circuits is also an important constraint used when synthesizing circuits using 2-by-3 switches (Y gates) discussed in chapter 3.
3. Every output of a reversible gate, which is not used to provide information to other gates in the circuit, or is not measured in quantum is called a *garbage signal* (*bit, qubit, etc*). Garbages waste energy in non-quantum technologies. They waste computing resources in quantum technologies, hence their name. They waste also energy in quantum computing when they are measured. Authors of related papers in literature call them also “waste”. A good synthesis method should minimize the number of garbage signals. In quantum technology, minimizing the number of garbage bits (garbage qubits) is one of the

main requirements of synthesis. The number of garbage qubits is related to the number of ancilla bits. The input signals in special circuits called “quantum oracles” are replicated to the outputs to be measured together with the decision qubit of the oracle (for instance in the famous Grover algorithm [Grover96]). These signals are not counted as garbage signals in the synthesis process. Thus there are different logic synthesis requirements on designing arbitrary reversible circuits and their subcategory of quantum oracles. This must be reflected in the synthesis methods, and will be illustrated in examples in next chapters.

4. Arbitrary logic function can be converted to a reversible function by adding small number of additional bits initialized to Boolean constants. These are the so-called ancilla bits and the synthesis goal should be to keep the number of ancilla bits as low as possible. Traditionally, most authors of papers in reversible logic reduce the number of ancilla bits to zero when possible (i.e. when the original function specification is reversible). I follow this tradition in new algorithms that I developed.
5. Every gate output in a reversible circuit can be used only once (the fan-out count of each output is equal to one). If two copies of a signal are required in a reversible circuit, a copying circuit must be used to reduce the power (in case of adiabatic-reversible CMOS technology). In quantum this is done because the technology does not allow for fanout necessary in a circuit. This copying circuit is a “Feynman gate” with one

input set to zero. Thus the circuit remains reversible but with the width of one more bit. (It is known that only basis states can be copied in quantum, which in fact results from the “No Cloning Theorem” – this property is however not affecting the type of circuits that we design here. The explanation can be found in [Nielsen00]).

6. Few authors of papers in reversible non-quantum technologies allow for a small fan-out, arguing that this may only slightly increase the power loss and that adding more gates may also increase power loss, so there is a trade-off. The answer on when to add small fan-out and when not to depends on physical properties of a particular reversible/irreversible technology. I am not interested however in these reversible/irreversible technologies as I want to abstract from all technology-related issues. For the purpose of formalizing my synthesis algorithms, I wanted in this dissertation to have a clear definition of what is and what is not a reversible circuit, even if this definition that I use is limited to a (large) subset of circuits. So in this dissertation we always assume that the fan-out of more than 1 is not allowed for reversible gates [Fredkin82, Toffoli80a, Perkowski01, Perkowski01a].
7. The reversible circuit resulting from synthesis will be acyclic, it will have no loops. Again, this is a traditional assumption taken by authors in reversible design, but recently (2007) several authors have worked on asynchronous reversible/adiabatic circuits that do not satisfy this

requirement. I am not interested in this dissertation in sequential reversible circuits with loops or combinational circuits with loops, so I keep the traditional definition used by most authors. Thus in this dissertation, loops will not be allowed in the circuit. As showed in [Kumar07, Kumar08] quantum automata can be realized also without loops, so my methods will still apply to this category of sequential quantum circuits.

The research on CMOS reversible realizations emphasizes reversible micro-pipelines [Athas, DeVos00, DeVos01] in which the same arrangement of transistors (a rough counterpart of a standard MOS gate) is repeated several times to realize a reversible gate (similar to standard CMOS where function  $f$  and its complement are realized) . Next, these structures are repeated in micro-pipelines of such gates. The size of every gate is thus large and the circuit's delay is increased, but the dissipated power is essentially reduced (frequency of operation is also sacrificed, which practically makes these technologies of restricted use). The main requirement of CMOS is therefore to make a gate suitable for regular layout based on abutting cells and short connections, thus allowing for a very limited routing in micro-pipelines, [DeVos00, DeVos01]. Also, among regular gates with inputs and outputs going horizontally and gates located vertically, some types of gate functions lead to increase of a gate

only in vertical direction, while some other functions increase internal layout of the gate in two directions, thus they are not scaling well. We are interested in gates that scale well. The challenges of realizing regular arrays in adiabatic CMOS and in quantum circuit technologies are very similar, thus many approaches presented in next chapters of this dissertation apply to both technologies.

Another difficulty of synthesis is mapping general Boolean functions, especially multi-output functions to the existing reversible gates such as Toffoli gate, Feynman gate, Inverter, and Fredkin gate. In addition, SWAP gate is only used in quantum technology and not used in any other reversible technologies. With the methods proposed in this dissertation, this difficulty of realizing multi-output functions is also solved by creating new gates that are matched to some systematic methodologies that take their origins from EXOR PLAs. The EXOR PLAs realize the “Exclusive-Or-Sum-of-Products” form of logic, called also ESOP [Mishchenko02]. Another way of realizing reversible cascades may be by using factorized ESOP circuits realized in two-dimensional regular arrays [Perkowski93a, Perkowski93b, Perkowski95b, Perkowski95, Sarabi94, Song93a, Song98]. One of contributions of this dissertation introducing new families of reversible gates and synthesis algorithms for them, see next section.

## 2.4. Families of Reversible Gates for arbitrary number of inputs and outputs

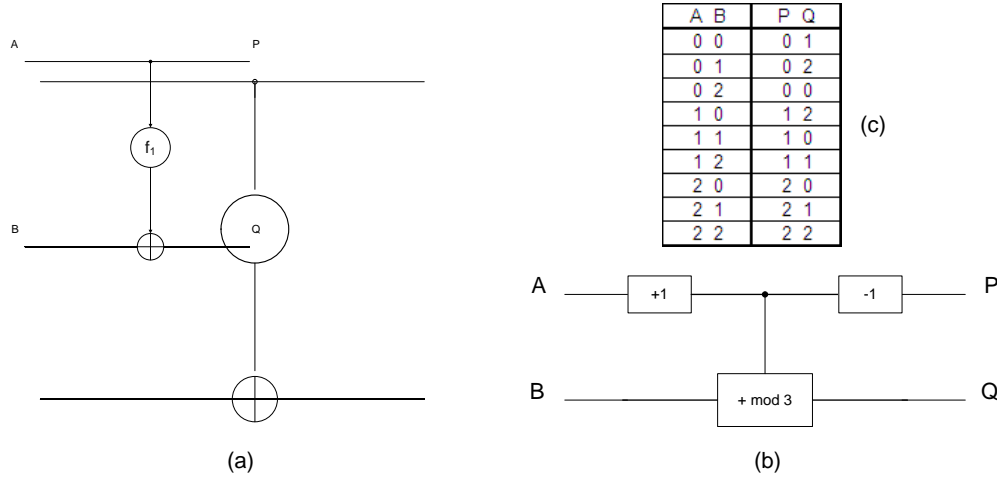


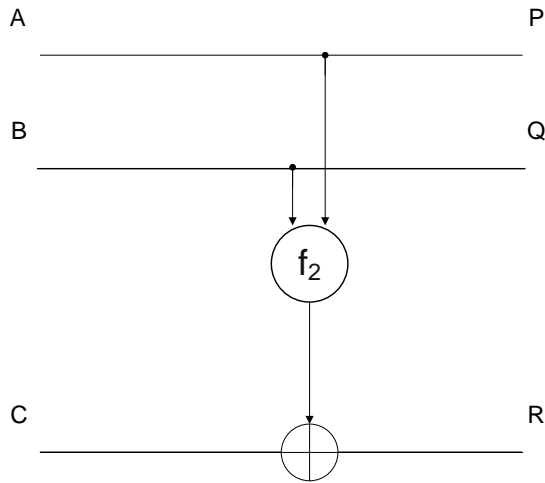
Fig. 2.1. Generalized Feynman Gate a) general schematics. b) an example in ternary logic. c) the truth table of the circuit from Figure 2.1b.

In this section, I generalize well known reversible gates such as Feynman and Toffoli. This generalization is done in the framework of Multiple-Valued Logic which is applicable to quantum circuits [Perkowski02]. Originally, I worked on developing synthesis algorithms for such Multiple-Valued Logic circuits. After the dissertation proposal I felt, however, that it is better to concentrate only on binary reversible circuits synthesis.

Figure 2.1a presents a generalization of the standard binary (2\*2) Feynman gate. Observe that wires A-P and B-Q can be binary (qubit) or

multiple-valued (qudit). Note the gate at the bottom is a Galois Field Addition (i.e. EXOR or GF(2) in case of binary) and  $f_1$  is an arbitrary binary or multiple-valued function. In case of binary logic, there exist only two functions of single variable  $A$  that can be used as  $f_1$ , wire (identity) and inverter (NOT gate). However, in case of multiple-valued logic there are very many functions of single variable (both reversible and non-reversible) that can be used as  $f_1(A)$ .

Function  $f_1(A)$  is always “added” (EXOR-ed in binary case) with  $B$  to produce the output of  $Q$  in Figure 2.1a. In ternary case, this is Galois Field (3) addition. Similarly, in quaternary case, this is Galois Field (4) addition, and so on. “Group gate” is a gate that satisfies the mathematical axioms of a group. Modulo additions and GF additions are examples of group gates. Any “group gate” can be used in mv case in place of Galois addition modulo addition especially increase the logic power of the above gate, which we call the “generalized Feynman gate”. One example of such gate is given in Fig. 2.1b and its truth table is shown in fig 2.1c. The block at the bottom of fig 2.1b is a modulo 3 addition. One of the generalization of EXOR gate. This is only one way of generalizing the Feynman gate. We call it the “Group-type generalization”.



*Fig. 2.2, Generalized 3\*3 Toffoli Gate size*

Figure 2.2 presents a generalized (3\*3) Toffoli gate. This is the main gate for synthesis in any logic pure or mixed (hybrid radix) and a universal gate assuming ancilla availability [Kerntopf01]. In the most general case, function  $f_2$  is an arbitrary binary or arbitrary multiple-valued function of two arguments. Observe that this function can be also hybrid, which means that its arguments A and B can have different radices. For example, A can be binary and B can be ternary. Again, function  $f_2$  is not necessarily reversible. It can be, in the most general case, an arbitrary irreversible hybrid function. Our theory works for all kinds of such functions, however, there still remains the question of which of these functions for  $f_2$  are efficiently realized in any particular technology, such as adiabatic CMOS or quantum circuits. There are  $16 = 2^4$  such functions of

binary variables, which is as many as the possibilities of putting symbols 0 or 1 to four cells of a Kmap that represents an arbitrary two-variable function. In case of ternary logic, the two-variable Kmap has  $3 \times 3$  cells, and each cell can have a 0, 1 or 2 which makes  $3^9$  functions. The number of such “generalized Toffoli gates” increases extremely quickly with the radix of logic. The EXOR operation in wire C is now replaced by any gate that has a group property (its operator table is the table of arbitrary group, i.e. satisfies all axioms of the mathematical group), such as Galois Addition or Modulo Addition. Similarly, for Feynman gate, we can create another generalization of this gate that is not using Galois operator, or even not using a Group operator. Concluding, one can state that very many gates can be created that generalize the familiar Toffoli gate, thus our methods from chapters 5 and 6 can perhaps be extended to generalized Toffoli gate because of the similarity of these gates to standard Toffoli. This will be however left as a future research area.

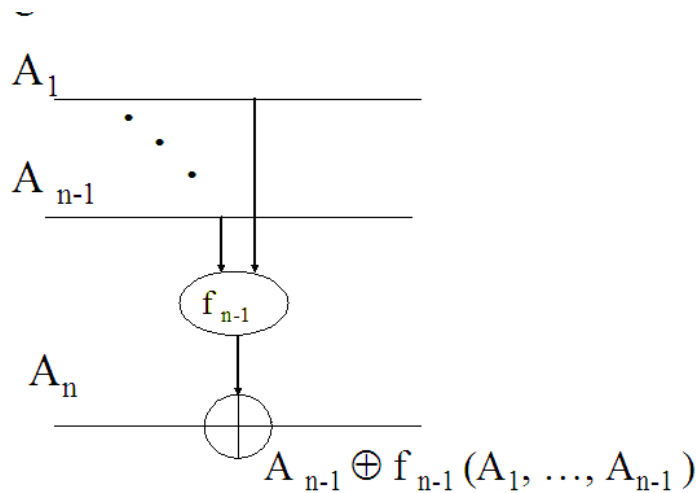
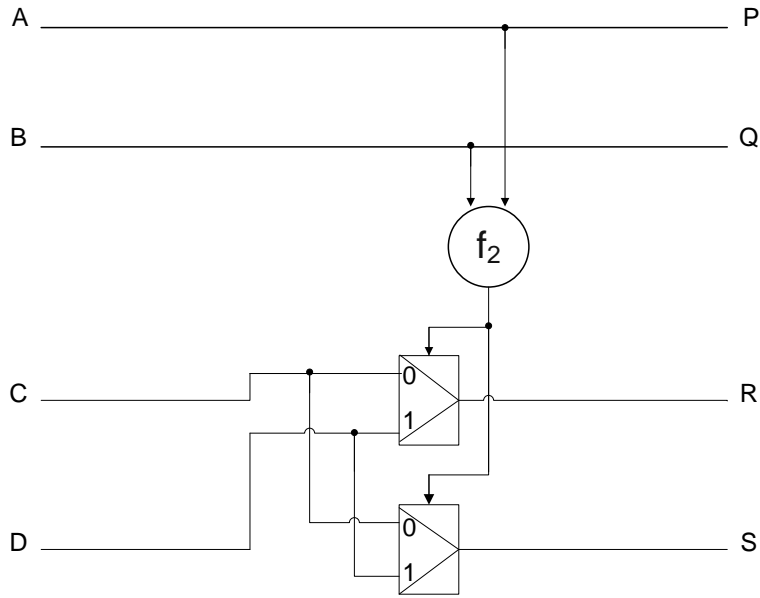


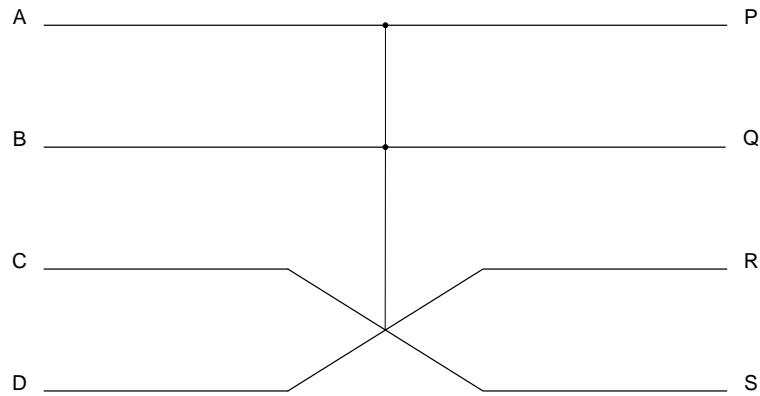
Figure 2.3. Generalized  $n \times n$  Toffoli Gate

Figure 2.3 presents a family of Toffoli gates for an arbitrary value of  $n$ . It can be easily verified using truth tables that all these gates are reversible. Observe again that all functions  $f_{n-1}$  can be binary, multiple-valued or hybrid.

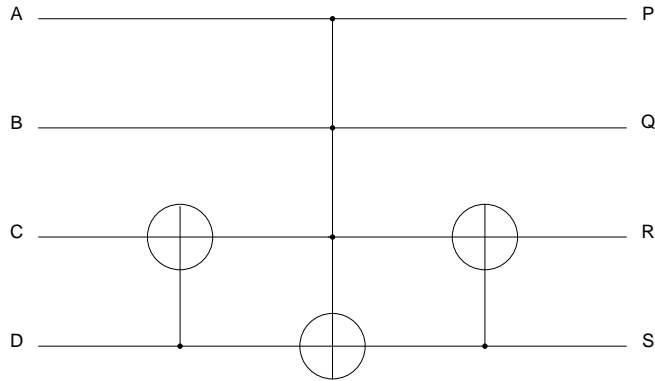
The concept of binary Fredkin gate can be generalized to a “Fredkin family of gates” with an arbitrary number  $n$  of inputs as follows:  $P_1 = A_1, P_2 = A_2, \dots, P_{n-2} = A_{n-2}, P_{n-1} = \text{MUX}(f_{n-2}, A_{n-1}, A_n), P_n = \text{MUX}(f_{n-2}, A_n, A_{n-1})$  where  $f_{n-1}$  is arbitrary function of  $n - 2$  variables (in general, binary or multiple-valued) being a control variable of the multiplexer, input  $A_{n-1}$  is a data input 0 and input  $A_n$  is a data input 1 of the multiplexer. This family has the same applications as the Toffoli family. The gates from this family may be, however, easier to realize in some technologies in which realization of the multiplexer is cheaper than realization of the EXOR. Again, this can be generalized to multiple-valued logic, and MMD-Like algorithms can be generalized and modified to be able to synthesize using these gates, as it was demonstrated in the Ph.D research of Maher Hawash [Hawash10].



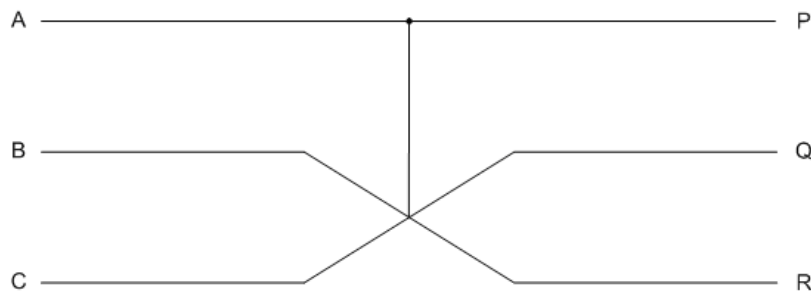
*Fig 2.4a. Generalized 4\*4 binary Fredkin Gate. This schematic emphasizes that a Fredkin gate which is a controlled-SWAP uses multiplexers to execute the SWAP. Here  $f_2$  is an arbitrary function of A and B used for control of the multiplexers.*



*Fig 2.4b. The schematics of 4x4 double-controlled Fredkin gate from Fig. 2.4a to explain controlled-SWAP.*



*Fig 2.4c. 4x4 double-controlled Fredkin gate realized using two Feynman gates and 4x4 Toffoli gate. This schematic emphasizes that Fredkin family of reversible conservative gates can be built using Toffoli family of reversible gates.*



*Fig. 2.4d. Standard 3x3 Fredkin gate as a controlled-SWAP.*

Figure 2.4a presents a 4\*4 Fredkin gate, Again, this figure is general, the wires (qubits) can have arbitrary different radices, f2 is an arbitrary non-reversible function with a binary output.

Figure 2.4b shows a double-controlled binary 4\*4 Fredkin gate, which operates as follows:

**If  $AB=1$ , then  $[R:=D; S:=C]$  else  $[R:=C; S:=D]$ .** It is thus double-controlled SWAP gate, its internal realization from CNOT and Toffoli gates is shown in fig 2.4c. Finally, the 3\*3 Fredkin gate is shown in Figure 2.4d. The control bit A is binary, while the bits B and C can be of arbitrary radix, but they must be of the same radix. This gate discussed in literature has always binary values of bits B and C.

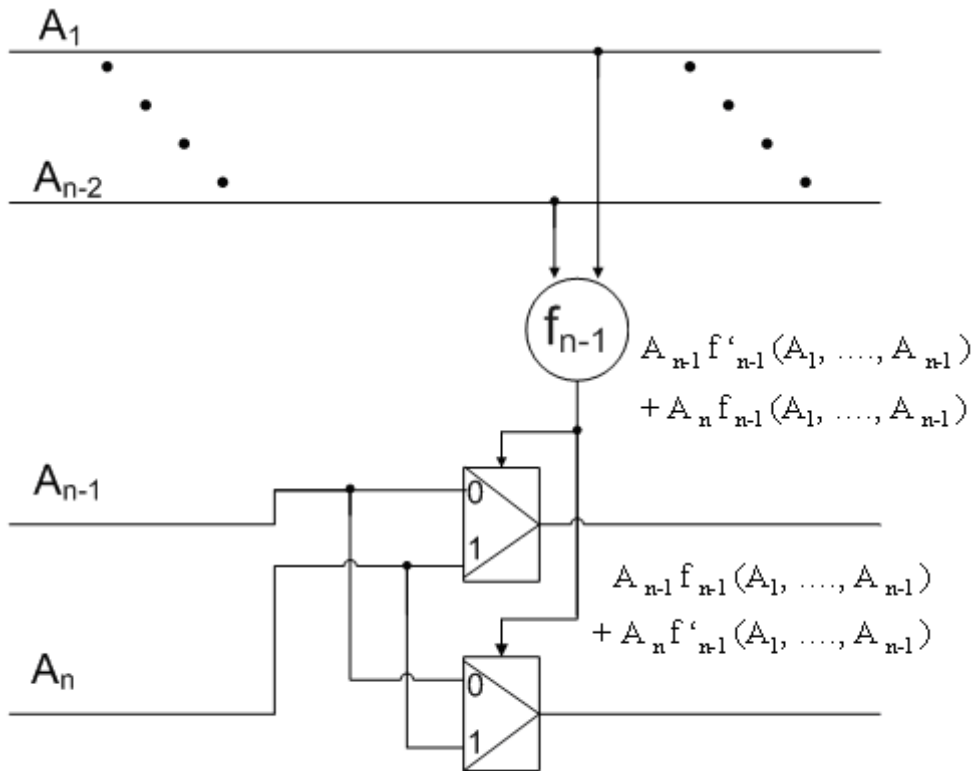


Fig. 2.5. Generalized  $n*n$  Fredkin Gate

Figure 2.5 shows a family of  $n \times n$  Fredkin gates. Kerntopf introduced a family of  $3 \times 3$  gates that have the maximum number of cofactors [Kerntopf00]. These gates were named Kerntopf gates in [Perkowski01c] and found useful for regular structures. The concept of Kerntopf gate can be generalized to a Kerntopf family with an arbitrary number  $n$  of inputs as follows:  $P_1 = A_1, P_2 = A_2, \dots, P_{n-2} = A_{n-2}, P_{n-1} = \text{MUX}(f_{n-2}, A_{n-1}, A_n), P_n = \text{DAVIO}(f_{n-2}, A_n, A_{n-1})$  where:  $\text{MUX}(x,y,z) = x'y + xz$ ,  $\text{DAVIO}(x,y,z) = x'z + y$ ,  $f_{n-2}$  is arbitrary function of  $n - 2$  variables (in general, binary or multi-valued) being a control variable of the multiplexer, input  $A_{n-1}$  is a data input 0 and input  $A_n$  is a data input 1. Davio gate (function, expansion) is named after Dutch researcher Davio who when worked for Phillips, created a complete theory of such expansions, invented these gates and characterized them mathematically.

There are no algorithms in the literature to synthesize with Kerntopf gates and very few algorithms to synthesize with Fredkin gates (all from PSU group only). There are also no algorithms to synthesize with generalized Fredkin gates or other MV generalizations than those used by Hawash.

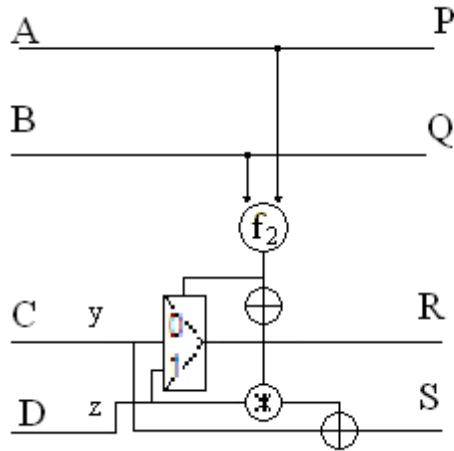


Figure 2.6. Generalized 4\*4 Kerntopf Gate.

The simplest generalized Kerntopf gate is shown in Figure 2.6. The reader may verify that each output is a balanced function and that the gate is reversible. The balanced functions have some interesting uses in general logic and reversible logic design.

In chapters 5 and 6, I will use only binary Toffoli family of gates. It remains as a future research to generalize all these algorithms to all other gates introduced in this section.

## **2.5. Explanation of Terms fundamental to this Dissertation: “Cascades” And “Relational”**

We call our reversible circuits “cascades” since they have no ancilla bits or very few ancilla bits. It is also typical in cascades that they are realized stage-by-stage:

- either from left to right (from inputs to outputs),
- from right to left (from outputs to inputs),
- Or from both sides to the middle (bidirectional search algorithms [Yang08]).

Thus, synthesis of reversible cascades, although not similar to classical logic synthesis methods, is most similar to the well-known traditional and not much used circuits known as “logic cascades”. They were introduced in 1970s by Tsutomu Sasao for Bubble Memory technology and are not used much in classical technologies. Moreover, the cascades presented by Sasao and subsequent authors were not reversible. These authors were however the first ones that introduced the permutation group theory to logic synthesis, and now it is the main mathematical tool used in reversible and quantum logic circuits.

The concept of designing reversible cascades with a small number of ancilla bits (or none) originates from our research group at PSU [Yen04, Khlopotne02]. My dissertation is one of the first documents to investigate this topic. The concept of reversible cascades

introduced by the PSU team, have been used by nearly all authors of algorithms to minimize reversible circuits. These other authors created software that was superior to PSU software, because they used better search algorithms than those proposed originally by the PSU team. Especially the program MMD [Miller03] and the program by Agarwal and Jha [AgarwalJha04] are very competitive and are treated popularly as landmark achievements in reversible logic synthesis. These programs are used by researchers and industries to design reversible and quantum circuits. Although better than my early programs and other PSU first programs from around year 2003, these improved programs by Miller et al and Jha et al still do not allow designing circuits of sizes that are made possible by modern technologies, because the possibilities of these programs are still limited. This is why new approaches were developed in this dissertation.

Another term in the title of this dissertation that still requires an explanation is the term “*relational input-output specifications*”. The concept of a don’t care is known in logic synthesis. The don’t care is when the output of a minterm (cell of a Karnaugh map) can be assumed in the synthesis process to be either a logic zero or a logic one. When we have two or more outputs in a circuit, however, certain constraints on the output sets of values are possible that cannot be formalized by the concept of don’t cares. For example, when two outputs of a logic cell in a Karnaugh Map are either 00 or 11 but never 01 or 10. This leads to the concept of the generalized don’t cares and Boolean Relations invented by Robert Brayton [Brayton89] and applied to decomposition of classical logic by Perkowski [Perkowski97].

In case of reversible logic, the concept of relational specification gets a new light. Obviously in case of reversible logic every existing circuit must execute a permutation of input/output vectors. This fact produces an additional constraint on the output values in the relational specifications. For instance, if there were two don't care cells in the K-map and values 00 and 10 have been already taken by mappings of selected input minterms then, the two remaining minterms initially mapped to don't cares, must be finally mapped to unique values. So, if we map the first of them to 11, the other can only be mapped to 01. We call this property "relational input-output specification" since certain relations (constraints) must always be satisfied because of the nature of reversibility. It is known that these kinds of relational specifications are useful in several problems of reversible/quantum circuit synthesis [Kumar07]. Specifically, relational specifications occur when engineers design quantum automata oracles for quantum algorithms, or parts of quantum oracles [Lukac02, Lukac02a, Lukac03, Lukac05, Lukac05a]. These parts of circuits may initially be specified as non-reversible. Such descriptions appear in many-level oracles, reversible and quantum state machines and other problems. This is an entirely new research subject of reversible logic synthesis, not discussed by other authors (in 2008 Manjith Kumar defended MS thesis on this topic at PSU, but he started his thesis only after I was already working on this topic) [Kumar08].

When completing relational descriptions to standard completely specified reversible functions, one has to take into account such properties of reversible functions as the

“balancedness”. This means that each single-output function, a component function of a  $k \times k$  reversible function must be the so-called “balanced function”. Balanced function is a function that has the same numbers of all its logic values. In binary case it means the same numbers of ones and zeros (true and false minterms in the K-Map). This property limits the search while selecting gates or while assigning values to the unspecified cells in relational specifications.

## 2.6. Background of previous Research Areas in Quantum Circuit Synthesis

There are three main areas of research in quantum circuit synthesis:

- 1) *Synthesis of truly quantum circuits that are not permutative circuits* (these circuits are described by unitary matrices that are not permutative). These circuits are used as components to synthesize permutative gates. Their quantum nature is used in creating cost functions to evaluate quality of the circuits synthesized by algorithms I developed. This approach to calculate costs is similar to classical design where the logic designer does not care about internal electric circuit design of gates from the logic cell library that he uses, but uses the characterization (cost) parameters of ready predesigned library cells such as the area or power consumption of the logic cell. We have no interest in this dissertation in the non-permutative synthesis, i.e. designing quantum primitives. In our algorithms we will just use the costs of gates from libraries created using methods shown in next chapters. Thus, in this dissertation we will be always designing with reversible gates, and not with non-permutative primitives such as the quantum primitives

(there are members of our PSU team that work on non-permutative synthesis; Lukac, Khan, Hossain, Shah, Rosenbaum, etc).

- 2) Synthesis of general purpose permutative quantum circuits (described by permutative matrices, they are mathematically the same as classical reversible circuits). This is the topic of my dissertation, which when started in 2004 was the second PhD dissertation worldwide intended to synthesize permutative quantum circuits. The first PhD on permutative logic synthesis was written by Dr. Maslov [Maslov03a, Maslov05c, Maslov03b] who introduced the popularly used MMD algorithm (next improved by Miller and Dueck). I found that this famous algorithm is based on very simple ideas and lacks any type of optimization other than the second pass of the so-called “template matching” optimization rules. This algorithm has always exponential complexity, but in cases that it can complete synthesizing the circuit (of not too large size) with respect to memory resources, it gives a warranty of convergence with a correct result. My heuristic tree search algorithms that I designed initially did not have this convergence property, unfortunately. My algorithms worked irregularly, sometimes with worse results than MMD and sometimes much better. But they were unacceptable as they were not always convergent. The algorithm was keeping searching and you do not know if it will ever find a solution or is looping. MMD had a clear convergence criterium and it was its main asset. I attempted therefore to design new algorithms that would combine the good properties of the my search algorithm and the

systematic greedy nature of MMD that gives provable convergence. The challenge that I set to myself was to design an algorithm for the same task as MMD, but that would allow to synthesize larger functions with smaller costs. Another challenge was to solve efficiently several other tasks, as seen in Table 1.2.1. The results of this approach will be discussed in chapters 5 and 6.

- 3) Synthesis of oracles, i.e. permutative circuits in which there is one “decision” output and all inputs are replicated at the outputs (“go-through” wires). Such circuits are also reversible  $(n+1)*(n+1)$  circuits “with one ancilla bit and all inputs replicated as outputs”. These are a special case of permutative circuits and they require special attention and specific methods for their synthesis. They are the subject of PhD dissertation of Sazzad Hossain in our research group at PSU [Hossain09]. It should be mentioned, however, that all my methods can be adapted to the synthesis of oracles as well, although my methods would perhaps be less efficient with respect to their run-times for such special case of circuits than the special methods from [Hossain09]. I do propose one method specifically for such oracle circuits. Advantage of circuits produced by method when applied to oracles is that there is no need to add mirror circuits when the oracle is formulated with repeated inputs and one functional (yes/no) decision output. Such oracles constitute, however, a limited subset of oracles.

Note, that in the existing research areas above I did not list research on logic synthesis for non-quantum reversible technologies. This is simply because there is no such research! There exist known circuits and examples of interesting designs for such technologies, but no logic synthesis theory or synthesis algorithms. In this dissertation I develop such methods for one of these technologies – the Y-gate, also called the “Prieze Switch” or the “2-by-3 switch”. My other methods can be also adapted to this technology with macros build using the new algorithm, called 2-3-S-A. [Perkowski10].

## **2.7. Main Categories of Previous Research on Reversible Logic Synthesis**

The algorithms for synthesis of permutative quantum circuits and classical reversible logic circuits investigated so far belong to few categories that will be listed and analyzed below.

### **2.7.1. Group theory methods**

Group theory methods belong to two categories: analytical and GAP programming. In analytical methods researchers want to find new group theory properties and theorems that can be used to synthesize reversible circuits. These methods were developed by Alexis De Vos et al and Guowu Yang et al. [DeVos02, DeVos01, Yang05]. These methods introduce very nice mathematical theory to characterize reversible gates and circuits and to synthesize them from a permutation vector, but algorithmically they are all

based on exhaustive search so they are not very practical by themselves. These approaches were applied to only 3 qubit and 4 qubit circuits. They cannot be applied to larger circuits. In my opinion, the group-theory based methods should be used in conjunction with good search algorithms and different representations of reversible functions. But, so far, the authors of these papers were not interested in the sophisticated search strategies that are investigated in the area of Artificial Intelligence (AI), with their respective heuristic functions and other heuristics. I tried and dropped these Group Theoretical methods for the reasons explained in next chapters.

### **2.7.2. Enumerative methods based on group theory**

Enumerative methods are so far all programmed in a programming language/system for group theory called GAP (researches of Guowu Yang and Alexis de Vos with co-workers). My criticism of these methods is similar to one given in section 2.7.1. Only the breadth-first tree search strategy was used in all papers because this strategy is easy to program in GAP.

### **2.7.3. Methods based on certain ordering heuristics.**

Methods based on certain heuristics for ordering were introduced by Miller, Dueck and Maslov in several papers starting in year 2003. These methods sort all minterms of a specification truth table from the smallest to the largest. Next they choose successive

gates to realize them, thus permuting from top to bottom of the truth table in such a way that the top of the table becomes only mappings of minterms to itself. Adding every gate permutes a small subset of not-yet sorted minterms from top. The most well known of these methods is the so-called MMD (Miller–Maslov-Dueck) method [Miller03, Miller03a, Miller05b, Dueck03b, Dueck03c, Maslov04], which will be improved upon in this dissertation. This program is popularly considered a landmark in reversible design so I determined that MMD is a good benchmark software set for my dissertation to compare with and improve upon.

#### **2.7.4. Methods based on evolutionary algorithm ideas.**

These methods are based on Genetic Algorithms and Genetic Programming ideas [Lukac03, Lukac02, Perkowski03, Khan04a, Giesecke06, Giesecke97, Rubinstein01, Spector99]. All these methods simulate Darwinian evolution with operators such as crossover and mutation and the selection function of the fittest chromosomes (individuals in the population). The methods use different data structures and parameters but they are basically similar. They are very general and easily adaptable to various synthesis variants or gates, but they are limited to 3-bit and 4-bit circuits so they share weaknesses of the group theoretical and exhaustive search methods. Recently even the authors of these methods add other types of search to the basic evolutionary search [Sazzad09, Lukac09]. To be competitive these methods require parallel processing which was not a subject of

my dissertation. The main advantage of these methods is that they are applicable to non-permutative quantum circuits or any technology.

### **2.7.5 Methods based on AND/EXOR logic and Reed-Muller expansions**

These methods were introduced in many papers of Perkowski et al, Agrawal, Jha, Mishchenko, etc. [Perkowski01, Perkowski01c, Perkowski01a, AgrawalJha04b, Jha06, Khlopotine02, Mishchenko02]. These methods have good potential according to my judgment based on literature and common sense. They all use concepts of AND/EXOR simplification rules and search methods and allow the reuse of software developed in the past for classical logic circuits. These methods allow one to synthesize much larger circuits than all other approaches investigated so far by other researchers in the field. In my dissertation I created algorithms to combine these approaches with my own ideas. The drawback of these methods is that they create ancilla bits in a number similar to the number of inputs. As my other approaches achieved the goals of dissertation I did not continue this research, however, to create software for approach with few ancilla bits, this is a research topic and is done by Alberto Patino [Patino10]. These methods can be combined in a several ways with methods proposed in this dissertation to create circuits with one or more than one but limited number of ancilla bits.

There are several types of AND/EXOR circuits. Positive Polarity Reed Muller (PPRM) are EXORs of products of positive literals (i.e.  $ab \text{ exor } abc \text{ exor } ac \text{ exor } d \text{ exor } 1$ ), Fixed

Polarity Reed Muller (FPRM) have all variables consistently in one polarity. ESOP are two level AND/EXOR expressions being exors of products with no constraint on polarity of variables.

## **2.8. Technology-related costs were often not addressed in Previous Synthesis Methods**

The old (and also some new) reversible gates discussed by other authors tend to abstract from the applied realization technology. It should be, therefore, well understood that some of these gates are only theoretical constructions that are not directly realizable in every technology listed above. We can call them “macros” that are built from many truly realizable primitives. For instance the Toffoli gate cannot be directly built in any existing quantum technology. The construction of an m-input Toffoli gate for  $m > 3$  is difficult in most reversible technologies. This gate is very costly when m is large. It requires:

- 1) Either  $m-2$  ancilla bits
- 2) Or hierarchical recursive structures [Barenco95]
- 3) Or the difficult to realize gates such as the “order-m root of NOT” gate (this gate is used only in quantum technologies such as Ion Trap and NMR).
- 4)

Thus, the assumption of using the m-input Toffoli gates in synthesis and calculating their number as the “gate cost” is not reasonable for several reversible technologies including quantum circuits and classical CMOS technologies. Unfortunately such gates are used as

primitives in most of the published papers on quantum circuit synthesis. If we have to use such a gate, we have to calculate its very high (technology related) cost, and use that cost as a component of the total gate cost of the circuit. We have to use thus the so-called "quantum costs" for comparison.

Consequently, the goal of this dissertation is to improve on this weakness of most of the published papers. Our goal is to develop a synthesis method that:

1. Uses gates or gate primitives only if they are technology-realizable. For example, in quantum, we only use gates realizable in a selected particular quantum technology and calculate pulse-related costs of gates (very accurate costs). Realistic gate libraries are assumed with costs of gates characterized.
2. Evaluates the synthesis results based on the technology realization costs of these gates and not abstract costs.

The costs that I introduced in this dissertation relate not to mathematical abstracts such as the number of (complex) gates but to the true hardware costs closely linked to the complexity of the technology realizable primitives [Lee06]. For example, in quantum technology, this is the number of elementary electromagnetic pulses. As my initial research showed, these gate costs differ from technology to technology, thus a circuit from only Toffoli gates that is minimal for one technology can be far from minimum in another technology where say, Peres and Feynman gates may be much cheaper. This requires defining various cost functions. In various technologies the costs may be

numbers of something totally different, for instance the number of electromagnetic pulses in NMR quantum technology and number of pass transistors in Adiabatic CMOS. It is thus very difficult to compare costs of the same circuit realized in various technologies (“cost in apples” versus “cost in oranges”) but two realizations of the same circuit in the same technology can be compared. Similarly, two different circuits can be compared using their costs in the same technology.

## **2.9. Efficient Search Algorithms for Reversible Logic Synthesis**

As seen in Table 1.2.1, the goals of my dissertation are to design theory and software to synthesize larger circuits, for more demanding design requirements, and for more types of reversible technologies. The ideas are related to efficient search, efficient representation of search and proving convergence.

Regardless of which reversible gates are employed, and the initial synthesis problem specification, every synthesis algorithm has to deal with a search problem of certain type (like a genetic search or a tree-search). In the case of reversible logic synthesis, the search space of this problem is of an enormous size. It can be compared to solving the Rubik’s Cube in minimal number of moves (rotations). But the sizes of permutation problems in our synthesis problems are larger than the Rubik’s Cube Permutation Problem. Both the

Rubik's cube and the permutative circuit design problems are the problems of finding the given permutation as a composition of a minimal number of "primitive" permutations that come from a certain "library of elementary permutations". In case of Rubik's Cube, the problem is to find the final cube permutation (the state of the Rubik's cube) from the initial permutation. In case of the reversible circuit the problem is to find the circuit specified as a permutation from the trivial (identity) permutations. The gates from the library of elementary permutations correspond to rotations of parts of Rubik's cube. All these are called the permutation decomposition problems and known to be very difficult to solve.

In classical logic design, the synthesis problems are reduced to the well-known "combinatorial" search problems such as the "set covering problem" to find the minimum-size covering of minterms with prime implicants. Unfortunately, formulated as stated above, the reversible circuit synthesis (permutation search) problems are less structured but more constrained than the well-known "combinatorial" search problems (constraint satisfaction problems) of classical logic design. We have thus to find new search methods for reversible logic. No research on this topic other than the Agrawal/Jha papers can be found in the literature.

It is well known that several basic problems in Computer Aided Design of standard logic circuits in AND/OR/NOT base are NP-hard.

Background. NP-hard are optimization equivalents of NP-complete problems. NP-complete problems require exponential complexity to find a solution but only polynomial complexity to verify the solution. An example of NP complete problem is the well-known combinatorial problem to find a coloring of a graph (such that every two neighbor nodes of the graph have different colors) with less than  $k$  colors or prove that such coloring does not exist. This is a decision problem. The NP hard problem would be to find the coloring of the graph with the minimum number of colors such that every two neighbor nodes of the graph have different colors. This is an optimization problem.

Note that the synthesis problem in reversible circuit synthesis is treated by specialists as more difficult than the case of synthesis of classical logic circuits.

Why is reversible logic synthesis so difficult?

- (1) Because of the permutative nature, there is no possibility of finding a general nicely decomposable regular structure like AND/OR. Also, no general circuit decomposition method exists for reversible circuits such as the Ashenhurst/Curtis decomposition [Perkowski97] that decomposes a circuit to smaller circuits. At least so far nobody found approaches like these.
- (2) In reversible logic, the basic gates are of AND/EXOR type rather than AND/OR type. Search problems are known to be much more difficult for AND/EXOR types.

(3) The Fredkin gate is similar to two combined multiplexers, thus it does not belong to AND/OR or AND/EXOR logic. Further, the multiplexer based synthesis methods cannot be easily adapted to Fredkin gates. The conservative gates that generalize the  $3 \times 3$  Fredkin gate have similar properties to Fredkin, as I will show in Chapter 3 of the dissertation. No search methods are known for such gates.

*(Background: A conservative gate preserves the number of “ones” (symbols “1”) between every input vector and its corresponding output vector).*

Thus, since search is the most important aspect of reversible synthesis, one can state that, so far, only the adaptation of the AND/EXOR based synthesis methods have proven to be successful in synthesis for reversible logic. Also, the methods based on “permutative group theory”, which are not similar to classical logic synthesis approaches, have been developed for reversible logic [DeVos00, DeVos01, DeVos02, Yang04, Yang05]. But so far, these search methods were applied only to small functional specifications. In this dissertation, I use both AND/EXOR methods and new methods to synthesize both conservative and non-conservative reversible circuits.

It is already popularly known that the complexity of synthesizing large reversible circuits exceeds considerably the complexity of designing classical circuits with the same number

of bits. Efficient and effective methods of synthesizing reversible circuits are therefore necessary. The research of previous graduate students at PSU (Dipal Shah, Sazzad Hossain, Martin Lukac, Anas Al-Rabadi [AlRabadi01b, AlRabadi01a, AlRabadi02a, AlRabadi02b], Normen Giesecke, Karen Dill [Dill97, Dill97a], Ugur Kalay, Chris Stedman, Akashdeep Aulakh [Akashdeep05], Eric Curtis, Manjith Kumar, Yen etc) as well as other researchers world-wide that collaborate with our Portland Quantum Logic Group (Mozammel Khan, Dong-Hwa Kim, Pawel Kerntopf, Tsutomu Sasao) and other researchers since 2000 have not been successful in the sense that none of the algorithms created was clearly better than MMD. In this dissertation such an algorithm, MP, is created and experimentally analyzed. We built therefore a practical CAD tool for the synthesis of reversible and permutative binary quantum circuits that can synthesize the largest known actual circuits.

## **Chapter 3: REALIZATION OF REVERSIBLE GATES IN VARIOUS TECHNOLOGIES OTHER THAN QUANTUM CIRCUIT TECHNOLOGY**

### **3.1. Introduction and goals**

Chapters 3 and 4 present various technologies to realize reversible gates, they will present:

- a) Constructions of binary gates in various reversible technologies known from literature on reversible circuit realizations (chapter 3)
- b) How most important binary gates can be built from other gates in these technologies (chapter 3)
- c) Quantum gates (chapter 4). We present well-known gates and their generalizations that are done by me and by other authors. These gates include gates that are mathematically equivalent to gates discussed in chapter 3. This creates uniform notation and a methodology that allows our synthesis methods to be applicable to all technologies (chapter 4).

Chapter 3 discusses main reversible binary gates in the following reversible technologies:

**T1.** Double-rail based on SWAP, NOT and Fredkin gates,

**T2.** Double-rail switch-reversible of Alexis de Vos based on CMOS on/off switches,

**T3.** Conservative reversible logic based on  $2 \times 3$  switches, with double-rail and single-rail variants (Y gate, Priesse Gate, etc in various technological realizations),

These are all “generic” technologies as the basic component gates considered for these technologies; SWAP, NOT, on/off switches and  $2 \times 3$  switches, can be realized in many particular technologies such as fluidic, CMOS, optical, etc. We are not planning in this dissertation to present technical details for any of these technologies. Fluidic gates and optical gates are very similar from the point of view of structures of switches, so there is no need to distinguish at this point if the gate corresponds to the flow of electrons or to the flow of photons, etc. We do not explain physical, electronic or geometrical (layout) aspects of gates. We concentrate only on the construction of switches as it is the base of logic equations which we use to derive synthesis algorithms.

Gates from chapter 4 that have their origin in quantum technologies, are similar but not identical to gates from chapter 3. In this dissertation, we will discuss their similarities, differences and how to build these gates from other gates.

In theory, all binary gates introduced in Chapter 4 can be built from gates from chapter 3. (By this, I mean mathematical functional composition and not their physical design. Physical design is an open problem that does not belong to this dissertation). Although I do not give formal proofs, from the construction methods I show, it is obvious that the statement in the previous sentence is true. My first interest is to prove constructability

and universality of all these gates. *Constructability* means that gates can be in principle constructed, as it is done in mathematics and logic theory papers. *Universality* of a set of gates  $G$  means that all logic functions can be constructed from the set of gates  $G$ . My main goal here is however more than constructability and universality – I want to find efficient realizations based on gates from set  $G$ , as attempted in this chapter 3 for the gates shown here. Although I have no proof that gates from chapter 3 are in any formal sense minimal, I believe that they are practically minimal (or that they are close to minimal). For some of the gates I show the best known constructions from literature that have not been improved by anybody in last 20 years, although perhaps many authors tried. Therefore these gates are perhaps best. Other gates I constructed by myself and have no proof of their optimality, but the possibility that they are minimal is high as they are similar to the known gates that are believed to have minimal costs.

One can ask, why don't I try to create minimal gates. Observe that proving mathematically or by exhaustive search the minimality of gates would be a very difficult topic in itself (see theses of Martin Lukac and Sazzad Hossain where exhaustive software was built). Nobody proved so far the minimality of Toffoli gate in quantum (in the sense of minimal number of the NMR electromagnetic pulses that realize these gates), although this design is used in most realized quantum computers and in most theoretical papers. Finally, even if some better internal realizations of Toffoli, Peres or other gates from chapters 3 or 4 were found, my methods that use these gates as “library gates” will be still useful. The internal construction of the gate would be thus improved; similarly as for

redesigning a NAND gate from p and n transistors in CMOS technology would not affect the synthesis methods from NAND gates. So the methods to synthesize arbitrary binary reversible circuits from gates of Chapter 4, which are presented in chapters 5 –6, can be in principle applied to any reversible technology. For the first three types of technologies listed above, the gates such as Fredkin, Toffoli and Feynman can be inexpensively realized, so the methods I introduced are a good match between the abstract model and the technological realization.

Observe that when the original specification of the function is reversible, the circuit of the second type can be built without ancilla bits. When the specification is not reversible, a reversible circuit with a small number of ancilla bits can be built according to method (b), using the methods from chapters 5 - 6. We did not invent so far a good method to realize circuits using only inverters and Miller gates or other similar “reversible majority gates”. Such constructions are possible, as proved using the group theory approach to reversible logic [Yang04, Yang05]. The trouble is that I did not find any efficient method to synthesize algorithmically in this way. Therefore in next sections of this dissertation, I will assume the first methodology and use  $k \times k$  Toffoli gates.

Concluding, the following statements are true:





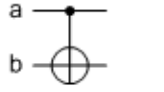

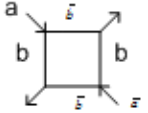
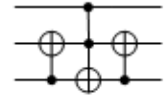
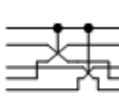



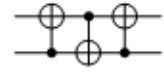

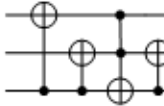

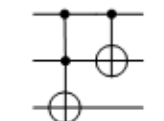
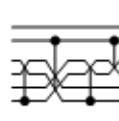
- (1) For every binary technology  $T_1 - T_3$  listed above, we can build Feynman, Fredkin, SWAP, NOT and Toffoli gates and their  $k \times k$  generalizations. These

gates will be discussed in this chapter. These gates have different costs in different realization technologies. Costs are not discussed in chapters 2 and 3.

- (2) The gates from point (1) can also be built in binary quantum technologies such as NMR and Ion Trap.
- (3) For quantum technologies from chapter 4, the multiple-valued and hybrid gates can be also built but such gates are not discussed in this dissertation.
- (4) We do not know if the multiple-valued gates can be built for the above four non-quantum technologies T1 – T3, and we will not work on this subject in the dissertation. There is much published about CMOS realization of non-reversible multiple-valued gates, but not much of this is in practical use. As many authors are critical about the future of multiple-valued circuits outside quantum technology, and also as I am not expert in circuit design, I do not plan to work on multiple-valued reversible circuits that are realized in other technologies than quantum.

Some of the gates from this chapter have not been built in all of these binary (including quantum) technologies, but only in some of them. In some cases the authors who are physicists do not give any explanation as to how they have designed the complex gates (like the Fredkin gate) from the primitive gates (for instance the 2\*3 switch gate). So I had to discover methods to synthesize these complex reversible gates in the above technologies from other reversible gates, and present all the gates here for the

completeness of my work goal of having my methods generally applicable in all reversible logics. The goal is to build the basic gates such as Feynman, Toffoli and Fredkin gate in all of the listed binary technologies in order to be able to calculate their costs in each technology. Next my unified logic synthesis methods will design the circuits using the same “generic” gates in all technologies, so the reference to the particular technology will become not necessary starting from chapter 5.

	c1 Quantum	c2 Double Rail Switch based	c3 CMOS deVos	c4 Double conservative
not				
Feynman				yes
Fredkin			yes	yes
Toffoli			yes	yes
SWAP 			yes	yes
Miller			yes	yes
Peres			yes	yes

*Fig. 3.1: Table comparing various basic reversible gates realized in quantum, double-rail switch-based, CMOS of DeVos, and double-rail conservative technologies.*

Comparison of gates for reversible technologies is given in Table 3.1. Bear in mind that a balanced function has half zeros half ones in each output and a conservative function preserves input value numbers, which means the number of values 1 is the same between inputs and outputs for every mapping. For instance, if the inputs are  $a = 0, b = 0, c = 1$  then outputs  $x, y, z$  should have exactly one in value 1 and two others in value 0. Similar property is true even if there are not equal numbers of inputs and outputs in reversible gates. Observe that such gates with unequal numbers of inputs and outputs are an extension of the concept of reversible logic given earlier in this dissertation. The requirement of having the same number of input and output signals is now relaxed. Examples are  $2 \times 3$  switch gates discussed below. These gates are conservative, as can be easily checked by the reader from the definition of conservative logic.

### **3.2. Double-Rail Logic Based on NOT, SWAP and Controlled-SWAP Gates.**

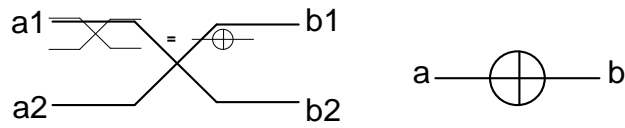
One of the methods for realizing reversible logic in hardware is the double-rail logic. There are several variants of this logic, even in the case of “adiabatic CMOS” technology. The gates in these variants differ in the number of clocks and in their internal gate designs. There are some technologies that the gate is represented not by two but by 4, 8 or even 16 duplicated gates, but here we are interested only in the double-rail logic concept. In this dissertation I will abstract from all electronic or physical details and I will illustrate only the main concepts of their switch structures, especially in double-rail

designs. In double-rail logic, a pair of two wires together correspond to a single wire in reversible logic. This is illustrated in Fig. 3.2.1 for the NOT gate. The circuit on the left uses two wires  $a_1, a_2$ , (i.e., the “double-rail”) to represent a logic value. Thus  $(1, 0)$  represents a logic of negated variable “a”, and  $(0, 1)$  the logic of positive variable “a”. The symbolic notation on the right shows my notation for single-rail (symbolic) notation of double-rail circuits:

$$\begin{aligned} a' &= (a_1, a_2') = (1, 0), \\ a &= (a_1', a_2) = (0, 1). \end{aligned}$$

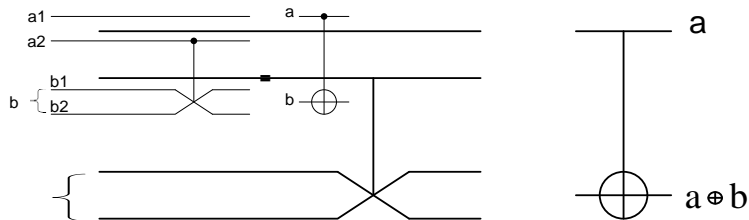
Figure 3.2.2 shows Feynman gate in double-rail and single-rail notations. The circuit on the left of Figure 3.2.2, is a double-rail realization of the Feynman gate at the right. Two wires on top left correspond to a single control wire on top right. The state 0 and 1 of these wires corresponds to value 1 of a. Two wires on bottom left correspond to the single wire b in the gate at right.

The swap of wires b1 b2 corresponds to the inverter (negation) gate (quantum notation symbol  $\oplus$ ) on bottom right in Figure 3.2.2. A controlled-SWAP in the left corresponds to Controlled-NOT in right. The control is with only one value. As we see in this example, which is a very good explanation of the principle of double-rail in reversible logic, the Fredkin (Controlled-Swap) gate in the double-rail logic corresponds to the Feynman gate in the single-rail (standard) reversible logic. This is because the SWAP gate in double-rail corresponds to an inverter in standard reversible logic.

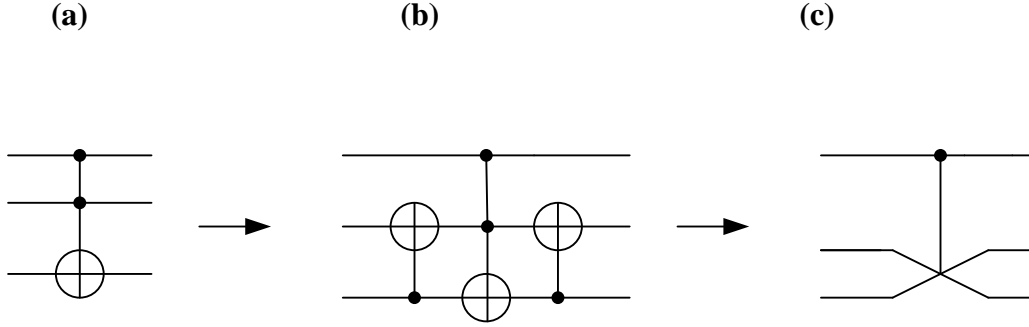


*Fig. 3.2.1.: Realization of NOT in double-rail technology. Every horizontal signal at the right corresponds to two physical wires.*

This observation that the NOT gate in single-rail logic is realized as the SWAP gate in dual-rail logic, is the base of many designs and the main principle of creating logic families. Figure 3.2.2 illustrates that if we can realize a Fredkin gate using any kind of gates or switches, then its counterpart Toffoli gate exists in double-rail logic.



*Fig. 3.2.2: Realization of Controlled-Not (Feynman) gate in double-rail logic.*



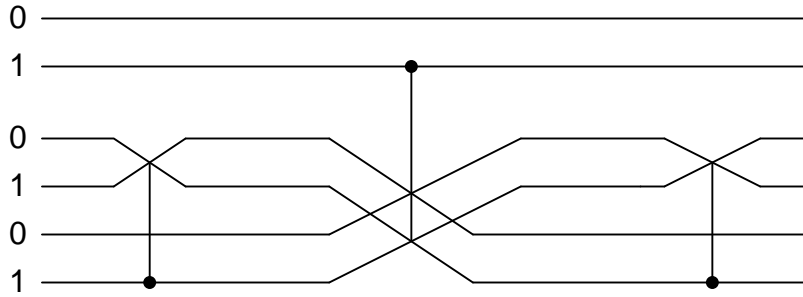
*Fig. 3.2.3: Realization of Fredkin gate using Toffoli and CNOT gates. By surrounding the Toffoli gate (Fig. 3.2.3a) with CNOT gates (Fig. 3.2.3b) we obtain the Fredkin gate (Fig. 3.2.3c).*

*Similarly the Toffoli gate can be obtained by surrounding the Fredkin gate using two CNOT gates. This way a Toffoli gate in double-rail logic can be realized, as shown in Fig. 3.2.4.*

Figure 3.2.3 shows that if we have Feynman gates we can build a Fredkin gate by surrounding Toffoli gate with two Feynman gates (as will be analyzed in full detail in next sections). Similarly, the Toffoli gate can be built from Fredkin gate by surrounding this gate by two Feynman gates.

Concluding, if we have an arbitrary  $3 \times 3$  reversible gate (in particular, an inexpensively realizable gate such as Toffoli or Peres gates) and many Feynman gates, we can create any other reversible gate of broad use. This is true also for quantum gates introduced in chapter 4. This method of realizing new gates is general and powerful, but the costs of the new gates are not necessarily the smallest ones for any given technology. This observation illustrates the power and importance of Feynman (CNOT) gate. This gate is inexpensive and it is very useful in all technologies from this dissertation. There are more

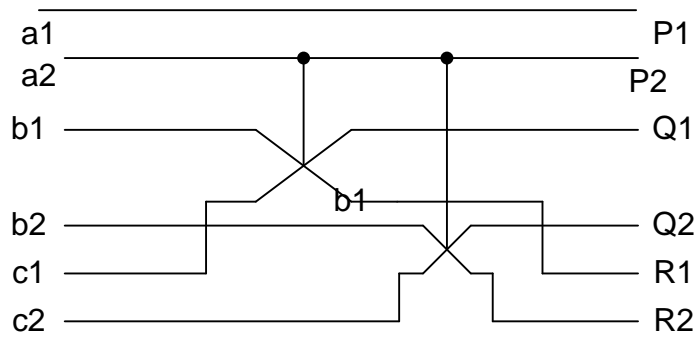
reasons why the CNOT gate is very important in all reversible technologies (and especially, in quantum technologies). Some of these reasons will be discussed in this dissertation as they relate to my synthesis methods.



*Fig. 3.2.4: Realization of Toffoli gate in double-rail logic using a double-rail Fredkin gate surrounded by two Controlled NOT gates.*

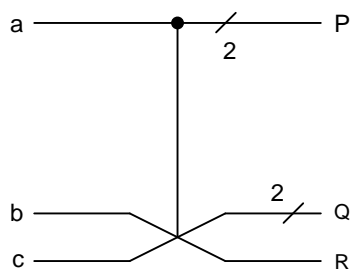
Concluding on Figures 3.2.1 – 3.2.4, we see that because NOT, CNOT and CCNOT gates can be realized in double-rail logic, every reversible logic circuit (reversible binary function) can be realized in double-rail logic. Thus the methods developed for quantum circuits in this dissertation can be applied to (at least) one variant of the double-rail logic presented in this section.

We can now calculate relative realization costs of NOT, Feynman, Toffoli and Fredkin gates in double-rail technology and next use these costs in our evaluation functions for circuits built in next chapters using “generic” gates.

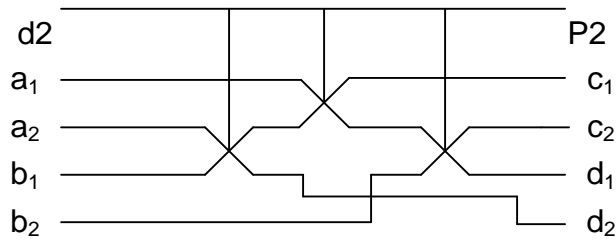


*Fig. 3.2.5.: Detailed circuit of double-rail Fredkin gate realized using two standard Fredkin gates controlled by the same signal a2.*

Observe that in Figure 3.2.5 every wire (signal) corresponds to a single Boolean value. The schematic in double-rail notation that corresponds to the gate from Figure 3.2.5 is shown in Figure 3.2.6. Every signal (line) in this schematics corresponds to two wires (double-rail) from Figure 3.2.5. This is denoted by slash symbol and number 2 in Figure 3.2.6.



*Fig. 3.2.6: The gate from Figure 3.2.5 in another notation, every signal corresponds now to two wires from Figure 3.2.5.*



*Fig. 3.2.7: Detailed circuit of double-rail “new gate” realized with three standard Fredkin gates.*

Figure 3.2.7 illustrates another way of using controlled swap gates to realize other gates than Fredkin. Observe that when each wire is treated as in single-rail technology the circuit is conservative, but when the Boolean values correspond to pairs of wires, the circuit is no longer conservative, but it is still reversible. When the signal  $d2 = P2$  has value 0 then we have:

$$(c1, c2) = (a1, a2)$$

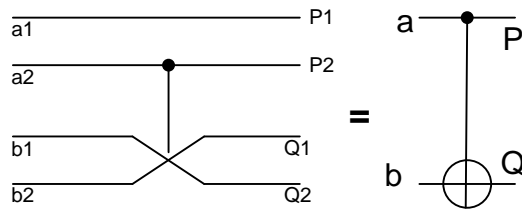
and

$$(d1, d2) = (\text{not } b1, \text{not } b2).$$

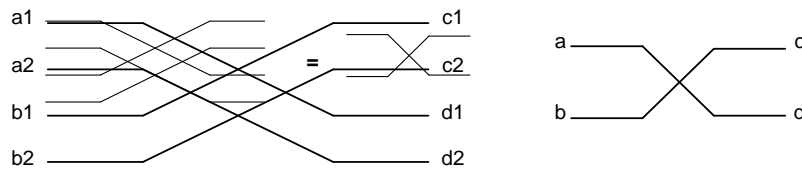
Thus for (double-rail) signal  $d = 0$  the (double-rail) signal  $c = a$  and the signal  $d = \text{not } b$ .

When  $d2 = P2 = 1$  then  $(d1, d2) = (a1, a2)$  thus  $d=a$ . When  $d2 = P2 = 1$  then  $(c1, c2) = (b1, n2)$  thus  $c=b$ . Then we swap between  $a$  and  $b$ . Observe that a functionality of a non-

conservative reversible gate was obtained without ancilla bits using only single-rail conservative gates. Observe that by composing conservative gates, we can only build conservative gates. Therefore, in single-rail logic, Feynman gate cannot be created from conservative primitives. However, in double-rail logic, every reversible gate is conservative with respect to all variables  $a_1, a_2, b_1, b_2, \dots$  etc. Therefore, non conservative gate such as Feynman, can be created from conservative single-rail gates. Above, we created a double-rail non-conservative gate.



*Fig. 3.2.8: Realization of CNOT in double-rail technology. Signal “a” is represented by  $a_1=0, a_2=1$ . Signal  $a'$  (negation of  $a$ ) is represented by  $a_1=1, a_2=0$ .*



*Fig. 3.2.9: Realization of SWAP in double-rail technology.*

Figure 3.2.1, Figure 3.2.8 and Figure 3.2.9 are basic gates from which we can construct all double-rail reversible gates. Repeating double-rail constructions we get quadruple-rail designs which have different clocking but the same logic construction principles. This dissertation gives examples of larger gates based on principles from chapter 3 and also of quantum gates from chapter 4. Observe that we used SWAP, NOT and Controlled-SWAP as our fundamental gates, but we did not specify how each of these gates is built: optically, electronically, in DNA, quantum dot, cellular or otherwise.

### **3.3. DOUBLE-RAIL OPEN/CLOSE SWITCH REVERSIBLE CMOS LOGIC OF ALEXIS DE VOS**

Alexis DeVos and his group have built several VLSI chips in (adiabatic) CMOS that realize reversible logic circuits. These chips are reversible both logically and physically, which means the logic input signals can be given to physical outputs and obtained as logic outputs on physical inputs and vice versa. Professor Perkowski's students simulated all these circuits and found decreased power consumption when compared with standard CMOS technology gates. The simulations proved the physical reversibility of all gates built by DeVos not only the logical reversibility. (a gate is physically reversible if it works correctly both ways from outputs to inputs and vice-versa). They have done similar work also for other adiabatic CMOS technologies, therefore I have a trust in the practicality of the concepts of these variants of reversibility.

In other words there are no clear distinction between input and output ports in De Vos technology, which is not true for some other reversible technologies in which the reversibility is only “logical” but not “physical”. All my methods from chapters 4 – 6 are for both variants of reversibility, but “physical reversibility” can find additional applications in circuit design.

### **3.3.1 Fredkin Gate**

We present first the Fredkin gate in this technology. The notation of Fredkin gate in Double-rail Switch-based technology of DeVos is shown in Figure 3.3.1. Realization of functions  $Q$  and  $Q'$  (symbols used in some papers as counterparts of  $Q_2$  and  $Q_1$ , respectively) is given in Figure 3.3.2. Symbol “ $a$ ” represents the naturally (statically) open and symbol “ $a'$ ” the naturally closed switch. Similarly double-rail functions  $R_1$  and  $R_2$  are realized, where  $R = (R_1, R_2)$ . This schematic is not shown. We will use both notations in this dissertation,  $Q'$  or  $Q_2$ , but some of them are more convenient in certain explanations. The notation with  $R$  and  $R'$  is confusing as it has no symbol for single-rail logic symbol corresponding to the pair of negative and positive values.

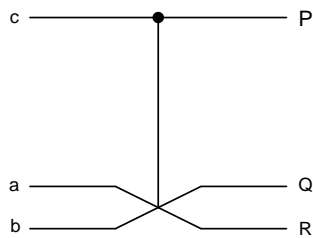
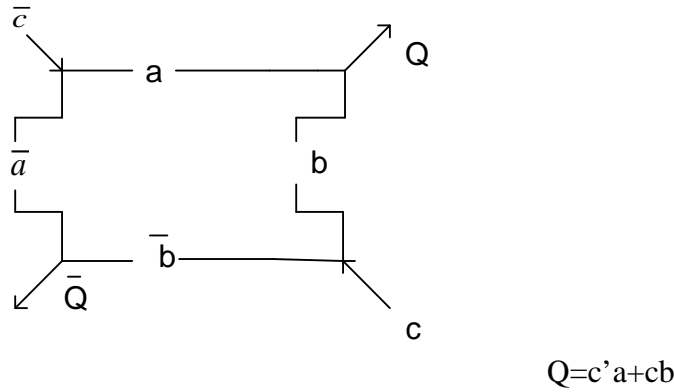


Fig. 3.3.1: Realization of Fredkin gate. This gate uses the notation for De Vos double-rail technology from Figure 3.3.2.



$$Q' = c' a' + c b'$$

Figure 3.3.2. Realization of function  $Q$  in Fredkin gate using De Vos technology. Similarly function  $R$  from Figure 3.3.1 is realized.

The function for  $R'$  ( $R2$ ) is the negation of the function realized for  $R$  ( $R1$ ). Observe that in Figure 3.3.3 there are two functions built from switches,  $F$  and  $F'$ , where function  $F$  is replicated twice, on top and on bottom, and function  $F'$  is also replicated twice, once on left and once on right. The same property is true in Figure 3.3.4, but with different functions  $F$ . This way, every function can be realized as reversible by EXOR-ing it with a variable. The exoring is done by the dual construction that uses a variable and its negation.

Observe however, that Fredkin gate from Figure 3.3.2 is different, as the upper branch is the negation of the left branch and the right branch is the negation of the lower branch. Both styles of De Vos gate design require repeated switches for variables and its negation.

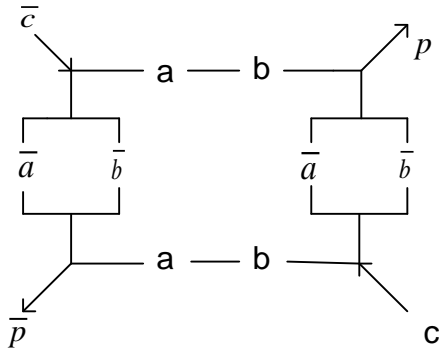


Fig. 3.3.3: Realization of Toffoli gate in double-rail DeVos technology. As we see, this is a kind of double-rail technology, with wires  $a$  and  $a'$  representing signal  $a$ , and so on for signals  $b$ ,  $c$  and  $P$ .

### **3.3.2 Toffoli Gate.**

The function realized in Figure 3.3.3 for output  $p$  is  $c' ab$  in upper branch controlled by  $c'$  and  $c(a' + b')$  controlled by  $c$  in the right branch. Thus the function realized for  $p$  is the following:

$$p = c' ab + c (a' + b') = c' (ab) + c (ab)' = ab \oplus c$$

Similarly, the function realized for negation of  $p$ , denoted by  $p'$  is  $c' (a' + b')$  (controlled by  $c'$ ) and  $(ab) c$  controlled by  $c$ . Thus the function for negated  $p$ , denoted by  $p'$  is the following:

$$p' = c' (a' + b') + (ab) c = c' (ab)' + (ab) c = [ ab \oplus c ]' = p' \text{ (negation of signal } p\text{)}.$$

In this way, any reversible gate can be built from switches as a function  $p$  and its complement  $p'$ .

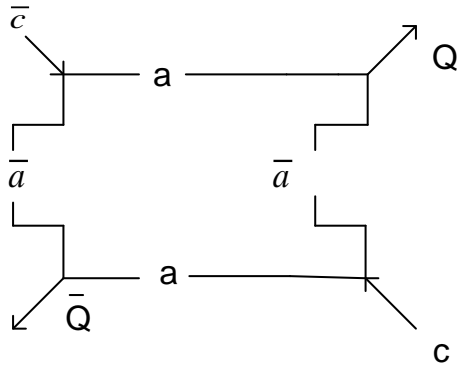


Figure 3.3.4. Realization of CNOT in De Vos technology.

### 3.3.3 Feynman Gate.

Realization of CNOT (Feynman gate) is shown in Figure 3.3.4. We can check that  $Q =$

$c'a + c a' = c \oplus a$ . Similarly signal  $Q' = c' a' + c a = (c \oplus a)$  which is  $Q'$  from definition.

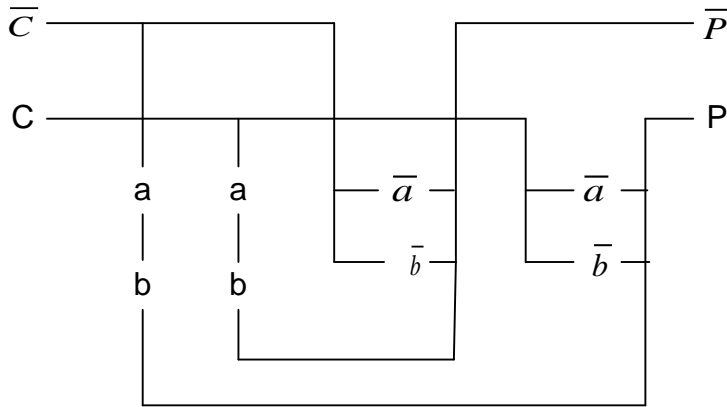


Fig. 3.3.5: DeVos CMOS circuit layout for pass-transistor diagram from Figure 3.3.3.

Figure 3.3.5 presents another way of drawing pass-transistor realization of 3\*3 Toffoli gate. This way of drawing the gate emphasizes regularity of design and the double-rail principle of realization. Similarly all other reversible gates can be drawn and next the whole schematic of the circuit can be drawn using this layout. The switches in De Vos technology are realized using CMOS switches, but they can be also optical, fluidic or other ON/OFF switches (closed/open switches).

De Vos logic implementation illustrates that we can realize an arbitrary reversible gate in binary CMOS technology. In particular, the Feynman gate, 3\*3 Toffoli, 4 \*4 Toffoli, 3

\*3 Fredkin, 4 \*4 Fredkin, and all other binary quantum gates from chapter 3. Thus the binary circuits that result from all methods from subsequent chapters can be realized using De Vos switch-based no-clock reversible circuits.

Observe that using DeVos open/close switches, I can realize double-rail CNOT. From two such CNOTs I can realize quadruple-rail Toffoli. Similarly other quadruple-rail gates can be built.

As we see, the De Vos technology can create gates CNOT and Fredkin for the Double-rail technology from section 3.2. Big gates have been also realized and simulated by de Vos and also simulated at PSU.

Concluding, De Vos technology allows to realize every reversible gate from “*open/close switches*”, and is a double-rail technology. Particularly, Fredkin, Feynman, Toffoli, Peres, Miller, Kerntopf and Margolus gates can be realized this way. Although it is more than 10 years from the first introduction of De Vos gates, and although the De Vos gates were shown by several studies to reduce power, they are not competitive and not used much commercially. But hopefully, when realized with other type of open/close switches, for example, in fluidic or optical technologies, these gates will become very power-competitive. This is a hope of researchers in Y-gate technology, for example. Additional advantage of De Vos reversible technology is very high testability of his reversible

circuits, as presented in papers of Hayes and Markov and the M.S Thesis of Jeff Allen at PSU.

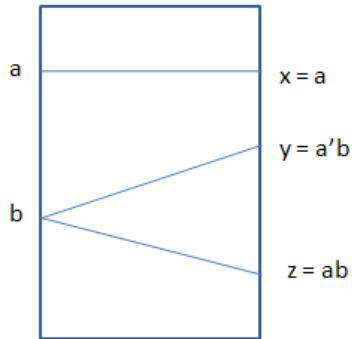
### **3.4. CONSERVATIVE REVERSIBLE LOGIC BASED ON 2\*3 SWITCHES**

The base of conservative logic (in this dissertation) is the “2 \* 3 switch” that can be realized in optical, fluidic, CMOS and several nano technologies. This gate has been popularized under the name of Y-gate by Forsberg [Forsberg04, Forsberg05]. Such switches were practically built, fabricated and tested.

There are some other  $n*k$  switches used in various technologies from which  $M*M$  gates can be constructed. We will not discuss them here; in general they are similar to  $2*3$  switches. As the terminology is not unified and the authors use the name switch, I proposed the name “ $2*3$  switch”, they are only provisionary for the purpose of this dissertation.

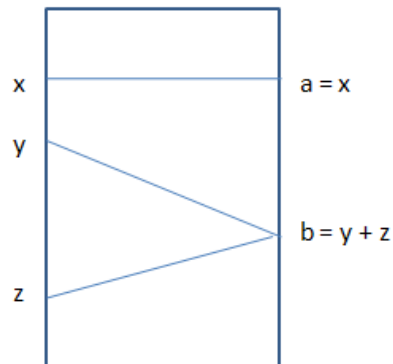
The sole building block of synthesis algorithm implemented in this section, is the Y-gate. Shown below in Fig. 3.4.1, a Y-gate is comprised of two inputs  $a$ , and  $b$ . The uppermost output,  $x$ , is simply the uppermost input,  $a$ , passed directly through to the output. The

middle output  $y$  is the product of the  $b$  with the negation of  $a$ . The lowermost output  $z$ , is the product of  $a$  with  $b$ .



*Figure 3.4.1: Y-gate*

The Y-gate is easily reversible, so what were formally the inputs now become the new outputs. Likewise, the former outputs become the new inputs. The inverse of the Y-gate is shown below in Fig. 3.4.2, with inputs notated  $x$ ,  $y$ , and  $z$ . The uppermost output,  $a$  is the uppermost input  $x$ , passed directly to the output. The lowermost output  $b$ , is the Boolean addition of inputs  $y$  and  $z$ .



*Figure 3.4.2: inverted Y-gate*

The derived truth table for the Y-gate, shown in Table 3.4.2, yields unique outputs vectors for each respective input vector, demonstrating the reversibility of the gate. A gate constructed from y-gates is referred to as single-rail, if it can be comprised solely from positive variable inputs. Double-rail gates are comprised from positive variable inputs and their corresponding negative variable inputs. Gates can only be realized in single-rail variant, if they are conservative. Therefore, from inspection of the Fredkin truth table in Table 3.4.1, the Fredkin gate is conservative, and therefore can be implemented as a single rail variant.

<b>a</b>	<b>b</b>	<b>c</b>	<b>x</b>	<b>y</b>	<b>z</b>
0	0	0	0	0	0
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	1	0
1	1	0	1	0	1
1	1	1	1	1	1

*Table 3.4.1. The truth table for Fredkin gate to be used in next examples.*

Table 3.4.2: Y-Gate truth table

a	b	x	y	z
0	0	0	0	0
0	1	0	1	0
1	0	1	0	0
1	1	1	0	1

Let us now appreciate some properties of a double-rail CNOT gate from Figure 3.4.3. Observe that the forward switch, shown at left of Figure 3.4.3 has its mirror inverse switch (reversed function  $3 \times 2$ ) as shown at the right of Figure 3.4.3. Fredkin gate in single rail is shown in Figure 3.4.4.

By drawing all KMaps with  $2 \times 4$ ,  $2 \times 3$ ,  $3 \times 4$  etc inputs/outputs that are reversible, we can invent all new switch-like reversible conservative non  $k \times k$  gates of this type. The question is, of course, if there are any technologies in which these new switches can be physically built. So this aspect is left as future work. It suffices, however, that the  $2 \times 3$  switches have been realized [Forsberg04, Forsberg05] and these are the base of the methods presented.

As the  $2 \times 3$  gates are reversible both logically and physically, the inverse switch is created by just sending inputs to outputs of the standard  $2 \times 3$  switch. Observe that this switch gate is not a  $k \times k$  gate but this gate is still reversible, unlikely to all other gates from this

dissertation that are  $k \times k$  gates. This switch allows realizing gates in double-rail logic (not conservative) such as CNOT, Toffoli, Peres, etc. Please analyze Figure 3.4.3. The switch allows also realizing conservative gates in single rail technology, as illustrated by a Fredkin gate from Figure 3.4.4. Understanding these two gates (Y gate and its inverse) is crucial for understanding all next gates and methods to synthesize them in this chapter.

The Y-gate switch is a very powerful gate as it allows realizing an arbitrary single-rail conservative function without ancilla bits. Next by taking function  $F$  in conservative technology and its negation together, we can create the double-rail realization of an arbitrary reversible function.

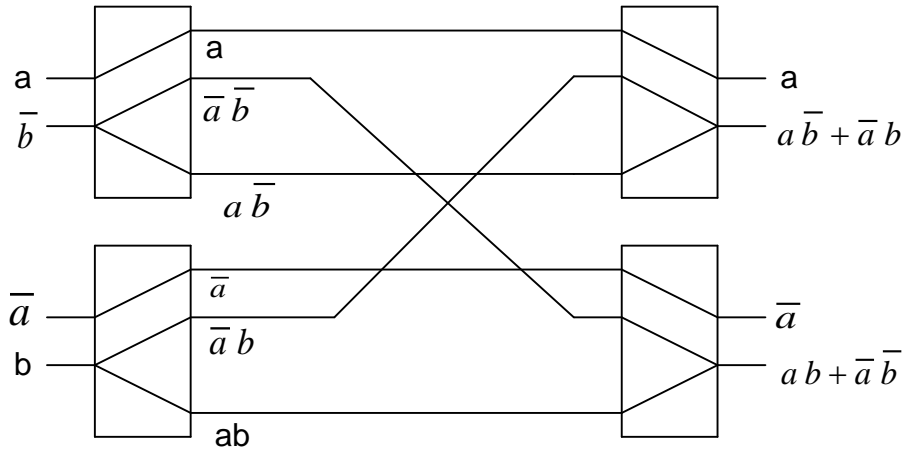
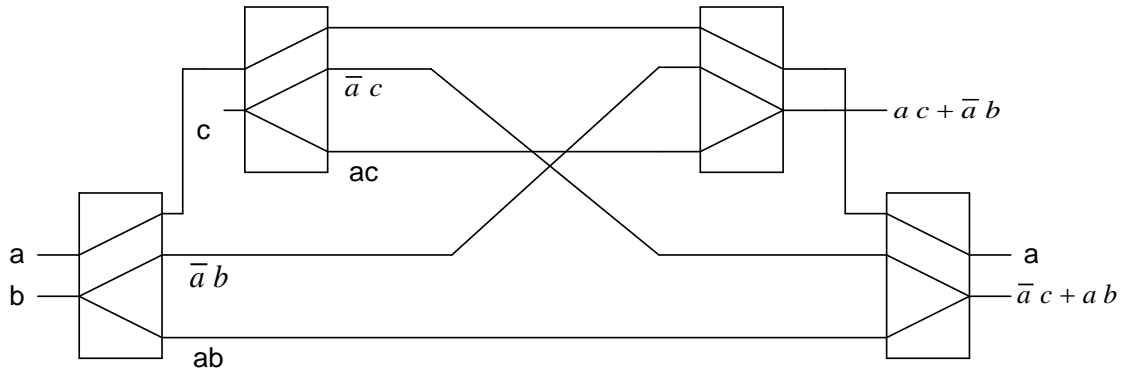


Fig. 3.4.3: Realization of CNOT in double-rail switch-based technology, using  $2 \times 3$  switch gates.



*Fig. 3.4.4: Realization of Fredkin gate in single-rail switch-based technology, using 2\*3 switch gates.*

As there was no paper explaining how the gates such as Fredkin are built with 2\*3 switches, and after trying to analyze these few existing solutions to extract some idea from them, a general method can be created based on a simple heuristic:

1. First we create a sequence of products of literals with more and more literals successively. We do this in all possible ways. This is illustrated in the left part of the upper row of gates in Figure 3.4.3. This way, functions  $a$ ,  $\bar{a}b$ ,  $ab$ ,  $\bar{a}c$  and  $ac$  are created. For gates of smaller size the number of such products is quite limited so my method is rather efficient. We cannot create fan-outs of any signals, including the primary inputs.

2. When we have the set of products available, next we can make sums of all possible pairs of disjoint products. Disjoint are products that share at least one variable that is in various polarities in them. This must be however done always in such a way that finally the number of primary outputs of the circuit that we create will be the same as the number of primary inputs.
3. We check that each function being a primary output must be balanced. (has half ones and half zeros).
4. Finally we check that the  $k \times k$  function created by us is reversible. (from definition of a one to one mapping).

This way we always create a  $k \times k$  gate. The above algorithm can be implemented as a tree search algorithm.

Because the switch gates and their inverses are conservative, every circuit composed by them that satisfies the above criteria is also conservative.

There are the following requirements:

1. The final gate must be of  $k \times k$  type for assumed number of  $k$ ,
2. No fanouts  $> 1$ ,

### 3. No cycles.

The heuristic idea of this algorithm is to create products first and sums later on, but we created some gates that mix creating the products and the sums. Creating sums of products of sums of products is thus also interesting. Other methods can be also created but they all make use of the fact that our requirements limit the sets of choices in both the first and the second stage of our method.

Using this algorithm yielded several conservative gates, one of them is shown below in Figure 3.4.3. Observe that there is a symmetry in all these circuit schematics (not a symmetry in the sense of Boolean functions). The symmetry is this: *For every switch gate there must be exactly one inverse switch gate*. This symmetry in general covers more circuit schematics than the symmetry that we see in Figures 3.4.1 and 3.4.2.

The new gate from Figure 3.4.5 is conservative and reversible. It is interesting as when some of its inputs are set to constants, it has outputs corresponding to products and sums, otherwise realizing multiplexers. In this way it is similar to the gates invented by Kerntopf and next improved by Zilic and Radecka. Inventing (or creating automatically) the set of all gates of this type for three, four or so primary inputs would be very interesting and can be left for future research. Another gate that was re-discovered by this method was the Fredkin gate, redrawn to the notation used by our method in Figure 3.4.7.

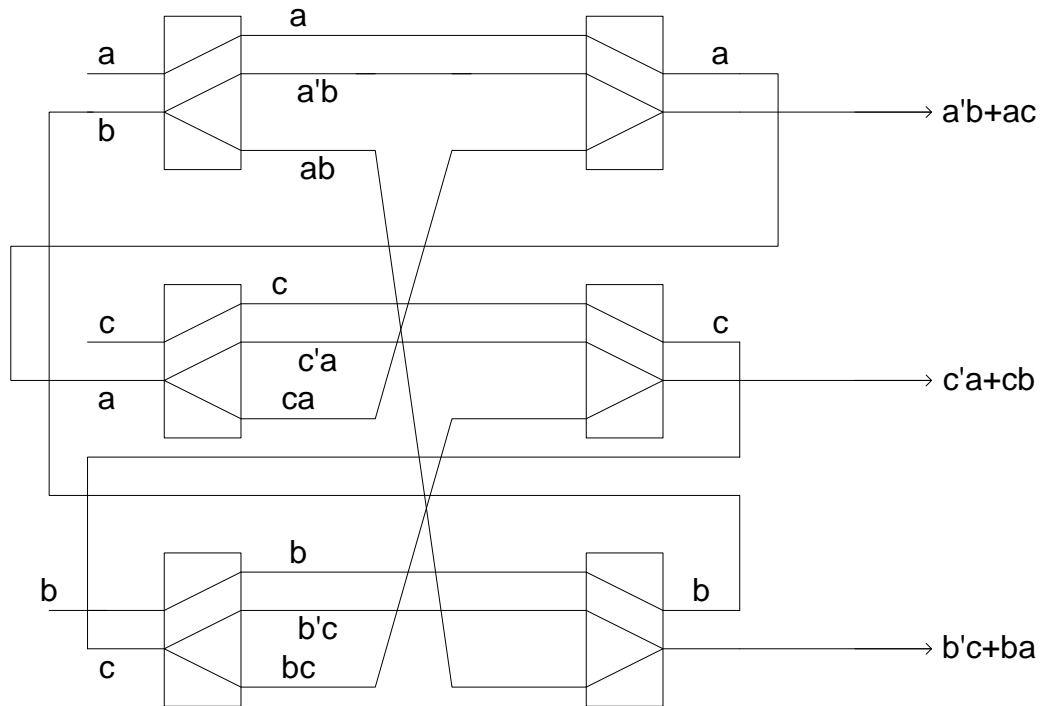


Fig. 3.4.5: Using Switch gates to build a new reversible conservative gate. Observe that in this circuit a loop of gates is allowed.

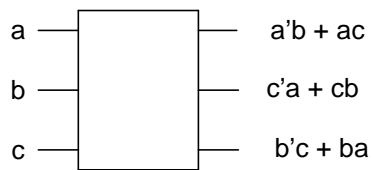


Fig. 3.4.6: Schematics of the new gate based on the schematics from Figure 3.4.5. This gate can realize products, inhibitions, sums and implication functions on its three outputs.

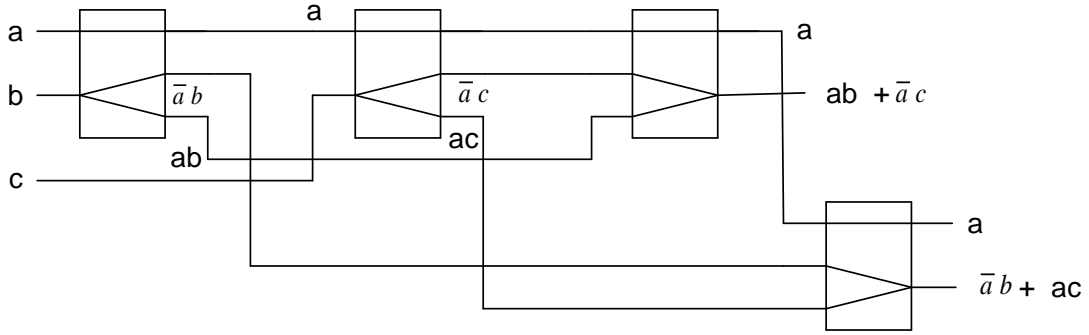


Fig. 3.4.7: Fredkin gate with ordering of gates created by the algorithm

We found that some quantum gates introduced as generalizations of basic quantum gates in chapter 4 can be also reinvented by this heuristic method based on composition of switch-based gates. The circuit from Figure 3.4.8 is built with just two controlled SWAPs, each built as in Figure 3.4.7. By verifying truth table of these functions we check that:

for  $a=0, b=0$  we have :  $R=c, S=d$ .

for  $a=0, b=1$  we have :  $R=d, S=c$ .

for  $a=1, b=0$  we have :  $R=d, S=c$ .

for  $a=1, b=1$  we have :  $R=c, S=d$ .

Thus a circuit schematic from Figure 3.4.7 can be drawn, which belongs to the family of generalized quantum gates analyzed and created in Chapter 2. To prove that this circuit belongs to the family from chapter 2 we rewrite it to the form from Figure 3.4.8. Next we perform algebraic transformations from Figure 3.4.9. This leads to the another equivalent schematic from Figure 3.4.9. But the reader may directly analyze from Figure 3.4.9 that this circuit is equivalent to the circuit from Figure 3.4.8.

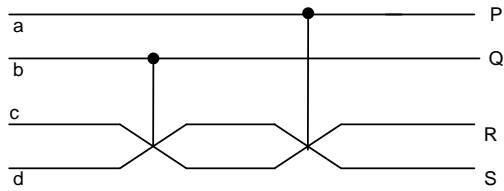


Figure 3.4.8. Schematic of a new single-rail reversible gate using Fredkin gates built from  $2 \times 3$  switches.

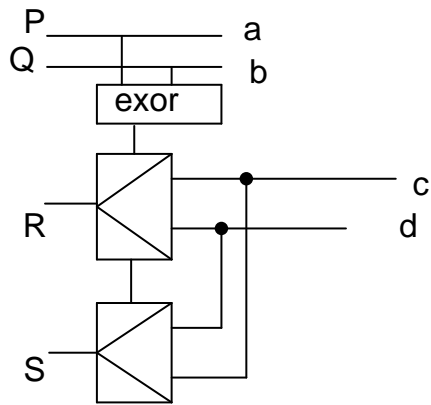


Figure 3.4.9. Another way of drawing the circuit from Figure 3.4.8 to emphasize its similarity to generalized gates introduced in Chapter 2. Both muxes are controlled by the Exor. This circuit, in contrast to circuits in other figures is drawn from right to left rather than from left to right to emphasize its reversible nature and to help using the analysis from Figure 3.4.10.

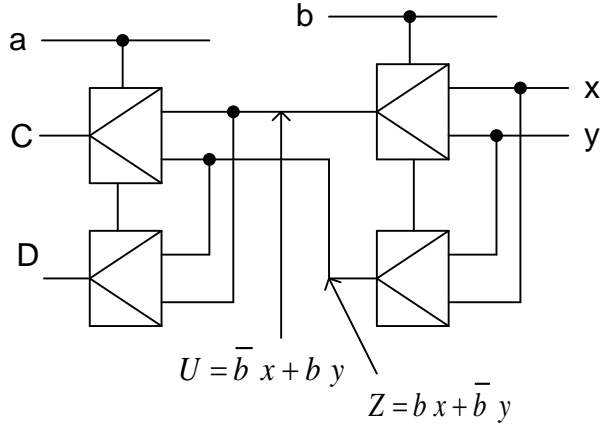


Fig. 3.4.10: Another notation for the circuit from two Fredkin gates given in Figure 3.4.8.

Assume  $x=c$ ,  $y=d$ .

$$\begin{aligned}
 \bar{a}u + az &= \bar{a}(\bar{b}x + by) + a(bx + \bar{b}y) \\
 &= \bar{a}\bar{b}x + \bar{a}by + abx + a\bar{b}y \\
 &= (\bar{a} \oplus b)x + (a \oplus \bar{b})y
 \end{aligned}$$

Fig. 3.4.11: Analysis of circuit from Fig. 3.4.10.

Concluding. Every conservative reversible gate can be built from 2\*3, 3\*2 switches and (hierarchically) from k\*k conservative gates built in turn from 2\*3 and 3\*2 switches. Next, by using double-rail technology, every reversible gate can be built from conservative gates. Thus, every circuit discussed in chapters 4 - 6 can (on the bottom of hierarchy of gates) be built from only 2\*3 and 3\*2 switches. The number of such

switches is the cost function of the solution which is used to compare various designs one with another.

### **3.5. SYNTHESIS OF REVERSIBLE LOGIC CIRCUITS BASED ON 2\*3 SWITCHES USING PROLOG**

Although the method presented in section 3.4 allowed me to create reversible circuits with Y-gates by hand, it was a heuristic approach which did not give any assurance if the circuit had a minimal cost. Therefore an exhaustive program was written in Prolog language that finds all solutions for small reversible functions.

Prolog language is used for circuit synthesis is elegantly explored in Tarau [Tarau07]. As Tarau clearly points out, the parallelism between digital circuitry and a logic programming language is inherent. The simplicity of Prolog initially appears restrictive, due to a limited repertoire of syntax available for problem solving. However with increased exposure to the language there is a simple elegance in being able to solve problems with a minimal tool set.

#### **3.5.1. PROLOG FOR EXHAUSTIVE LOGIC SYNTHESIS**

All programming was performed using SWI-Prolog/XPCE 5.6.64. The software is available for free download from the SWI-Prolog website at <http://www.swi->

[prolog.org/download/stable](http://prolog.org/download/stable) The code utilizes SWI-Prolog libraries, and therefore cannot be guaranteed to function as desired on other versions of Prolog. Prolog is a logic programming language. While it is procedural, its uniqueness from many other languages is derived from the fact that it is declarative, in that logic is expressed as relations. Execution of the declarations is carried out by means of queries.

Prolog is composed of a single data type, a *term*. Relations between terms are defined by *clauses*. Prompted with a given query, Prolog searches for a resolution expostulation of the negated query. If a query is can be expostulated, instantiations of variables that satisfy the query are found. Capital letters denote variables, while lowercase letters denote constants. In Prolog there are two types of clauses; rules and facts.

An example of a rule is **apple :- (fruit).**

This is interpreted as ‘fruit is true if apple is true’.

A fact is a clause with an empty body, such as: **fruit(apple).**

A fact can also be conveyed as a rule, such as: **fruit(apple) :- true.** Given this rule, a query can be given inquiring if apple is a fruit. **?- fruit(apple).**

The query would return a ‘Yes’, as apple is defined as fruit. A query can also be made to ask what things are fruits.

**?- fruit(X).**

**X = apple.**

A Feynman gate will be used to demonstrate basic gate functionality in Prolog. The Feynman gate has two inputs,  $a$  and  $b$ , and two outputs  $x$  and  $y$ .  $x$  is passed the value of  $a$ , while  $y$  is the exclusive disjunction of  $a$  and  $b$ .

The Feynman gate can be defined in Prolog by naming the clause after the gate, *feynmanTruthTable* and presenting the input and output ports as arguments, essentially creating a truth table. Constants 1 and 0 represent logic high and logic low respectively.

**feynmanTruthTable(0,0,0,0).**

**feynmanTruthTable(0,1,0,1).**

**feynmanTruthTable(1,0,1,1).**

**feynmanTruthTable(1,1,1,0).**

From the fact declarations, a query can be contrived to find an output vector, based upon a specified input vector. In the example below, an input vector of  $a = 1$ , and  $b = 1$  is used to find the value of the corresponding output vector represented by variables  $X$  and  $Y$ .

**?- feynmanTruthTable(1,1,X,Y).**

**X = 1,**

**Y = 0.**

As expected, the query returns the only possible output vector,  $X = 1$ , and  $Y = 0$ , that satisfies the specified input vector. A gate can also be represented as an output vector that is a function of an input vector.

**feynmanFunction(A,B,X,Y):-**

**X = A,**

**Y is xor(A,B).**

Within the rule, the output  $X$  is set equal to input  $A$ , while  $Y$  uses the Prolog defined *xor* Boolean operation to define its relationship based upon inputs  $A$  and  $B$ . Queries for the function based Feynman gate implementation, can be made in the same manner as queries for the truth table defined Feynman gate.

A list in Prolog is simply a sequence of any number of items, for example **[apple, banana, blueberry, cantaloupe]**.

The first item in a list is referred to as the *head*, while the remaining portion of the list is referred to as the *tail*. While lists can be displayed as items between brackets separated by commas (e.g.  $[a,b,c,d]$ ), they can also be based upon the head and tail of a list such as **[Head|Tail]** .

An empty list in Prolog is denoted as empty brackets: [ ].

These syntax rules allow for multiple interpretations for displaying a list. A list such as [a,b,c] can also be displayed as [a|bc],[a,b,[c]], or [a,b,c|[]].

### **3.5.2. EXHAUSTIVE SEARCH ALGORITHM**

The algorithm for synthesis can largely be broken into three sections; gate generation, wire connectivity, and verification. Code fragments are included in this section to demonstrate key algorithmic concepts realized in Prolog.

#### **3.5.2.1. GATE GENERATION**

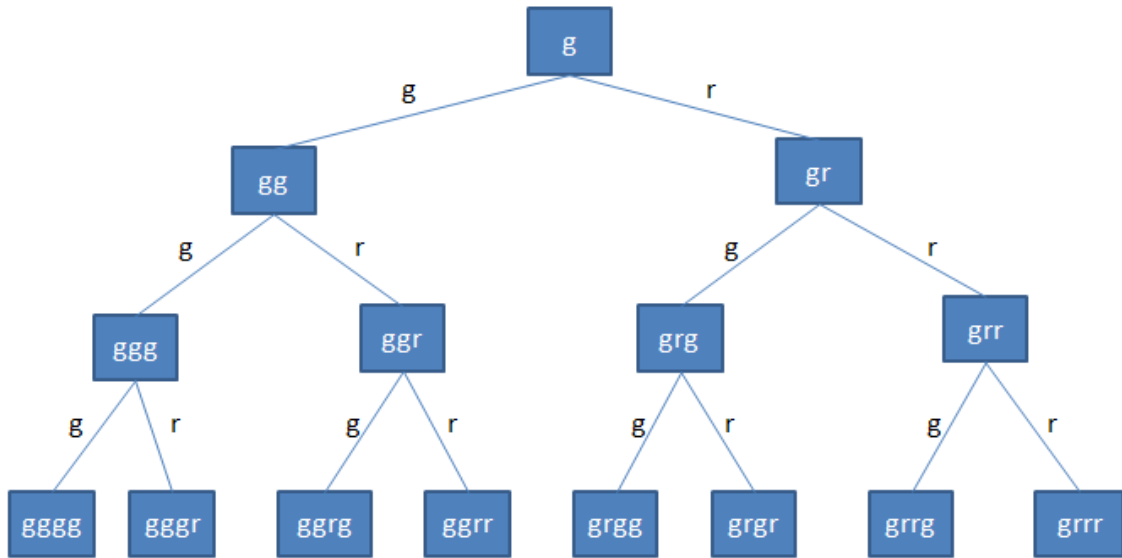
For the synthesis of reversible gates, the Y-gates may be used either in a typical manner, as depicted in Fig. 3.5.1, or in an inverted manner, as depicted in Fig. 3.5.2. Y-gates used in a typical manner will be referred to as *G-gates*, as the number of output ‘grows’ or increases with respect to the number of inputs. Likewise, Y-gates used in a reversed manner will be referred to as *R-gates*, as the number of outputs is reduced with respect to the number of inputs.

To construct desired reversible gates, outputs from certain Y-gates must feed into the inputs of other Y-gates. An order for how the ports of Y-gates are connected is established using a generator to create all possible combinations of gate orderings. A user defined limit is established for the number of gates to be implemented.

There are also a number of limitations that apply to how the sequence of Y-gates must be ordered:

- (1) The sequence must begin with a G-gate, and must conclude with a R-gate.
- (2) There must be equal number of G-gates and R-gates.

The gate generation is constructed by creating a tree. As the first gate must be a G-gate, the head of the tree is a lone G-gate. A binary tree is formed as each level of the tree creates two nodes from each previously existing node. One of the new nodes will have a G-gate added in sequence behind it, while the other new node will have a R-gate added in sequence behind it. The resulting tree is depicted below in Fig. 3.5.1. Each node on the tree is a list, and the tree is a list of nodes, making it a list of lists.



*Figure 3.5.1: Y-gate generation tree*

To explain in further detail, as previously discussed, a single G-gate, the root of the tree is the first list. The root list of each tree, in this case the lone gate is the entire list and the head, is taken and concatenated with a G-gate to make one child node, and concatenated with an R-gate to make another child node. These children nodes are added to a new list. The root list is then traversed, so the concatenation process that occurred with the original root, now takes place with the next element in the list. This procedure takes place until the entire list (node) has been traversed. Once the entire list has been traversed, the same concatenation procedure is applied to the new list of nodes that were previously considered children.

The code for generation of the binary tree is shown below, with the first line serving to assure the appending process halts when the list is empty. The first argument of the binary tree determines the number of Y-gates to be used. The clause *append* is a library defined clause that concatenates the first two arguments of the rule to define the third argument.

```
binarytree(0,[],_,[],[]):-!.
binarytree(N,Left,Root,Right,L):-
    N1 is N-1,
    append(Root,[g],Left),
    append(Root,[r],Right),
    binarytree(N1,_,Left,_,L0),
    binarytree(N1,_,Right,_,L1),
    append([Left|L0],[Right|L1],L).
```

As previously established, there must be an equal number of G-gates and R-gates. This check is easily implemented by an established counter for the number of G-gates, and a counter for the number of R-gates. Clause *count*, counts the number of occurrences of its first argument in its second argument (a list), and stores the count as its third argument.

```
applyconstraint([],[]).
```

**applyconstraint([H|T],L):-**

**count(g,H,N1), %count g**

**count(r,H,N2), %count r**

**applyconstraint(T,L0),**

**is\_equal(N1,N2,H,L0,L).**

### **3.5.2.2. WIRE CONNECTIVITY**

In the implemented Prolog program, the inputs to the Y-gates are no longer just single bits, but lists. Boolean functions are defined that perform the desired operation on every item in a list. Use of the Y-gates and reverse Y-gates requires the Boolean functions of negation, conjunction and disjunction.

The argument *Mask* is a bit-string of all '1's, used for exclusive disjunction with another sting, to obtain the negation of that string.

**applyF(and,Input1,Input2,Output):-**

**Output is  $\wedge$ (Input1,Input2).**

**applyF(or,Input1,Input2,Output):-**

**Output is  $\vee$ (Input1,Input2).**

**applyF(negate,Mask,Input,NegatedInput):-**

**NegatedInput is xor(Mask,Input).**

The Y-gate and reverse Y-gate are constructed using Boolean functions, as well as the clause *different*, which elsewhere defined, ensures its two arguments are not equivalent. In this implementation, it specifically ensures the outputs are correctly ported to.

**applyF(ygate,Mask,A,B,X,Y,Z):-**

**X = A,**  
**applyF(negate,Mask,A,NA),**  
**applyF(and,NA,B,Y),**  
**applyF(and,A,B,Z),**  
**different(Y,0),**  
**different(Z,0).**

**applyF(rygate,Mask,A,B,X,Y,Z):-**

**A = X,**  
**applyF(or,Y,Z,B),**  
**applyF(ygate,Mask,A,B,X,Y,Z).**

Input and output lists are comprised of unused Y-gate inputs and output ports. While the program iterates through placing together all feasible connections, the used ports of the Y-gates are removed from their respective unused port lists.

In the code for the wire connection below, the *special\_delete* clause deletes the input from the input list or the output list, and updates whichever list was taken from.

#### **Grow Function:-**

**apply(g,Mask,Control,Input,Y,Z,InputList,OutputList,[X|NewInputList],[Y|[Z|NewOutputList]]):-**

```
    union(OutputList,InputList,InOutList),
    member(Input,InOutList),
        special_delete(Input,InputList,OutputList,NewInputList,NewOutputList),
        applyF(Ygate,Mask,Control,Input,X,Y,Z).
```

If arguments *Input*, *Y* and *Z* are ignored, as they are solely utilized for circuit printing purpose, the function shown above can be represented in the form:-

**apply(g,Mask,Control,InputList,OutputList,NewInputList,NewOutputList)**

*InputList* and *OutputList* represent the wire connectivity before applying *Ygate*, and *NewInputList* and *NewoutputList* represent the wire after applying *Ygate*. *Mask* is simply passed to *Ygate*.

#### **Reduce Function:-**

**apply(r,Mask,Control,Y,Input1,Input2,InputList,OutputList,[X|InputList],[Y|New  
OutputList]):-**

```
    member(Input1,OutputList),
        delete(OutputList,Input1,OutputList0),
        member(Input2,OutputList0),
        delete(OutputList0,Input2,
            NewOutputList),
    applyF(rygate,Mask,X,Y,Control,Input1,Input2).
```

Ignoring arguments *Y*, *Input1*, *Input2* , as they are solely used for printing purpose, the function is of similar form to the previously described Grow function.

**apply(r,Mask, Conrtol,InputList,OutputList, NewInputList,NewOutputList)**

The *member* function is used to select an element, while *delete* is used to delete an element from the list.

As double-rail implementations require negation of input variables, inputs, variable *HI* in the code below, are taken and XORed with a bit string of ‘1’s to create the negated input variable, *NHI*.

**doublerail\_inputs(\_,[],[]).**

**doublerail\_inputs(Mask,[H1|T1],[H1|[NH1|L0]]):-**

**doublerail\_inputs(Mask,T1,L0),**

**applyF(negate,Mask,H1,NH1).**

The above function is simply not incorporated into the synthesis of the single-rail reversible circuits.

The *syn* function, shown below for the double-rail implementation, forms the heart of the synthesis algorithm. The *applyP* function, defined elsewhere, takes the gate tree (pattern) and attempts to satisfy the end goal.

**synDR(NbOfVariables,NbOfYGates,Goal):-**

**N is (((NbOfYGates-(NbOfYGates mod 2))/2)-1),**

**get\_pattern(N,[g],PatternList),**

**all\_ones\_mask(NbOfVariables,Mask),**

**vars\_to\_bitstring\_ints(NbOfVariables,InputList0),**

**doublerail\_inputs(Mask,InputList0,InputList),**

**member(P,PatternList),**

**applyDP(P,Mask,InputList,[],Goal),**

**print\_variables(97,Mask,InputList0).**

The single-rail synthesis function is identical, with the exception of the omission of the *doublerail\_inputs* function, and the declaration of *N*, which establishes the input pattern to be mirrored.

### **3.5.2.3. VERIFICATION OF REVERSIBLE CIRCUITS**

As the implemented algorithm generates almost all tangible wire connections for all tangible gate combinations, many, and in most cases, the vast majority of contrived connections will be undesirable.

If synthesis is carried out without the objective of verifying a specified reversible circuit, all possible implementations are synthesized in the synthesis clauses, *synSR* for single rail, and *synDR* for double rail, below. The first argument of the function is the number of input variables. The second argument is the maximum number of Y-gates plus one to be used for synthesis.

The final argument is a variable used to implement previously discussed gate generation restrictions.

***synSR(NbOfInputs , NoofYgates , Goal).***

***synDR(NbOfInputs , NoofYgates , Goal).***

If synthesis is carried out with the objective of verifying a specified reversible circuit, only circuits will be implemented that meet the specified criteria (*Defined\_Goal*).

The same clause with a different argument

**synSR(NbOfInputs , NoofYgates , DefinedGoal).**

**synDR(NbOfInputs , NoofYgates , DefinedGoal).**

can be called. In this case, the argument, *Defined\_Goal*, invokes the additional functionality of comparing the synthesized circuits against a specified truth table, and thus only selecting viable candidates.

The argument *Defined\_Goal* is the decimal representation of the desired binary string output of the reversible circuit.

### **3.5.1. RESULTS OF EXHAUSTIVE SEARCH**

Successful circuit synthesis has been carried out for:

- (a) Single-rail Fredkin gate,
- (b) Double-rail Fredkin gate,
- (c) Double-rail Feynman gate,
- (d) Double-rail Toffoli gate.

A single-rail Fredkin gate is used to explain numerical arguments, and query results from the synthesis algorithm. Outputs for a single-rail Fredkin gate are defined as functions of their inputs:  $x = a$ ,  $y = ac + a'b$ ,  $z = ab + ac'$ .

For the synthesis,  $y$  is selected as 53, and  $z$  is selected as 83. These values are decimal representation of binary bit strings [1,1,0,1,0,1] and [1,0,1,0,0,1,1] respectively. The code for the decimal to binary bit-string conversion was taken from Tarau [Tarau07]. As output  $x$  is solely a control gate, it does not need to be specified as an argument, as it will be generated.

As a Fredkin gate has three inputs, the first argument to the function is three. As the Fredkin is comprised of four gates, the second argument to the function is three. The final argument is the numerical values of the bit strings.

A resulting query for a gate with  $y = 53$  and  $z = 83$  can be entered as follows.

**?- synSR(3,3,[53,83]).**

One of the resulting implementations to the query is listed below.

**[rygate I1=15, I2=48, I3=5, O1=15, O2=53]**

**[rygate I1=15, I2=80, I3=3, O1=15, O2=83]**

**[ygate I1=15, I2=85, O1=15, O2=80, O3=5]**

**[ygate I1=15, I2=51, O1=15, O2=48, O3=3]**

**c=85 b=51 a=15**

**true .**

The result is the gate sequence, which is read from bottom to top as GGRR. Identical numerical values for different ports of imply that all those ports are connected. As  $a$  and  $x$  are all 15 ([1,1,1,1]), this represents the control signal being passed sequentially first through the control of each non-inverted y-gate, then through the control of each inverted y-gate. As the last line of the answered query indicates, the input sequence to the circuit is:

$a = 15$ , or binary 1111

$b = 51$ , or binary 110011

$c = 85$ , or binary 1010101

To ensure these numerical input values yield the desired outputs, the results are easily verified through Boolean arithmetic.

$$y = ac + a'b$$

$$ac = \quad 1111 \quad a'b = 110000$$

$$\begin{array}{r} \underline{1010101} \quad \underline{110011} \\ 0000101 \quad 1100000 \end{array}$$

$$ac + a'b = 0000101$$

$$\underline{1100000}$$

$$y = \mathbf{1100101} = 53$$

Likewise,  $z = ab + a'c$

$$ab = \quad 1111 \quad a'c = 1110000$$

$$\begin{array}{r} \underline{110011} \quad \underline{1010101} \\ 0000011 \quad 1010000 \end{array}$$

$$ab + ac' = 0000011$$

$$\begin{array}{r} \underline{1010000} \\ \mathbf{z = 1010011 = 83} \end{array}$$

As the achieved output values are identical to the desired outputs, this serves to demonstrate the correct functionality of the implemented single-rail Fredkin gate. The resulting synthesized single-rail Fredkin gate is shown below in Fig. 3.5.2.

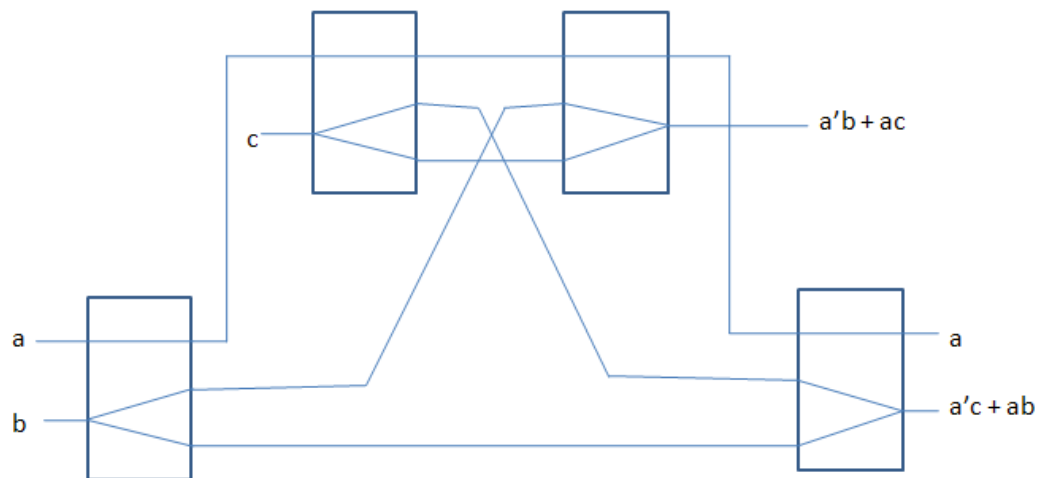


Figure 3.5.2: Single-rail Fredkin gate implementation.

One realization for a double-rail Fredkin gate is shown below, with the visual depiction illustrated in Fig. 3.5.3.

?- synDR(3,8,[53,83,172,202]).

[rygate I1=240, I2=5, I3=48, O1=240, O2=53]

[rygate I1=15, I2=80, I3=3, O1=15, O2=83]

[rygate I1=15, I2=160, I3=12, O1=15, O2=172]

[rygate I1=240, I2=10, I3=192, O1=240, O2=202]

[ygate I1=240, I2=85, O1=240, O2=5, O3=80]

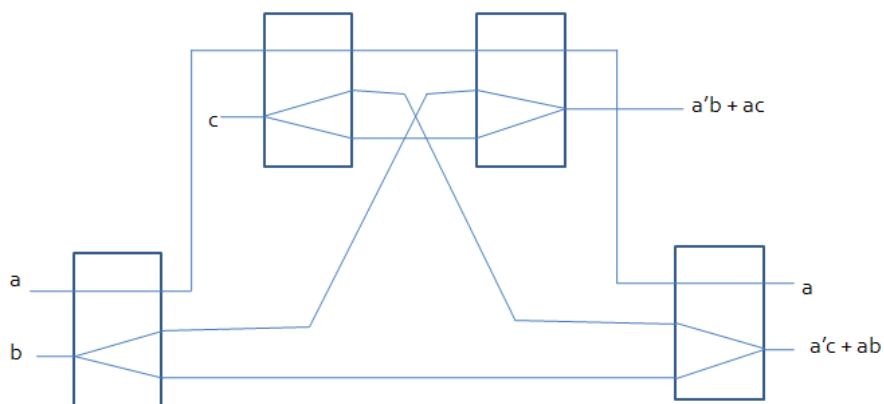
[ygate I1=15, I2=170, O1=15, O2=160, O3=10]

[ygate I1=15, I2=51, O1=15, O2=48, O3=3]

[ygate I1=240, I2=204, O1=240, O2=12, O3=192]

a=15 a'=240 b=51 b'=204 c=85 c'=170

true.



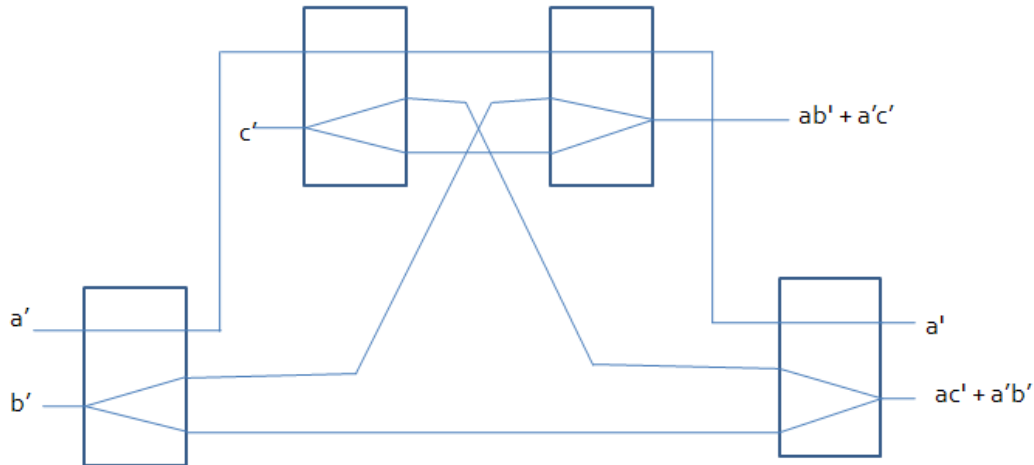


Figure 3.5.3: Double-rail Fredkin gate implementation.

One realization for a double-rail Feynman gate is shown below, with a illustrated depiction shown in Fig. 3.5.4.

[rygate I1=12, I2=2, I3=4, O1=12, O2=6]

[rygate I1=3, I2=8, I3=1, O1=3, O2=9]

[ygate I1=3, I2=5, O1=3, O2=4, O3=1]

[ygate I1=12, I2=10, O1=12, O2=2, O3=8]

a=3 a'=12 b=5 b'=10

One realization for a double-rail Toffoli gate is shown below, with a illustrated depiction shown in Fig. 3.5.5.

?- synDR(3,8,[86,169]).

[rygate I1=240, I2=6, I3=80, O1=240, O2=86]

[rygate I1=15, I2=160, I3=9, O1=15, O2=169]

[rygate I1=204, I2=1, I3=8, O1=204, O2=9]

[rygate I1=51, I2=4, I3=2, O1=51, O2=6]

[ygate I1=51, I2=10, O1=51, O2=8, O3=2]

[ygate I1=204, I2=5, O1=204, O2=1, O3=4]

[ygate I1=15, I2=85, O1=15, O2=80, O3=5]

[ygate I1=240, I2=170, O1=240, O2=10, O3=160]

a=15 a'=240 b=51 b'=204 c=85 c'=170

true.

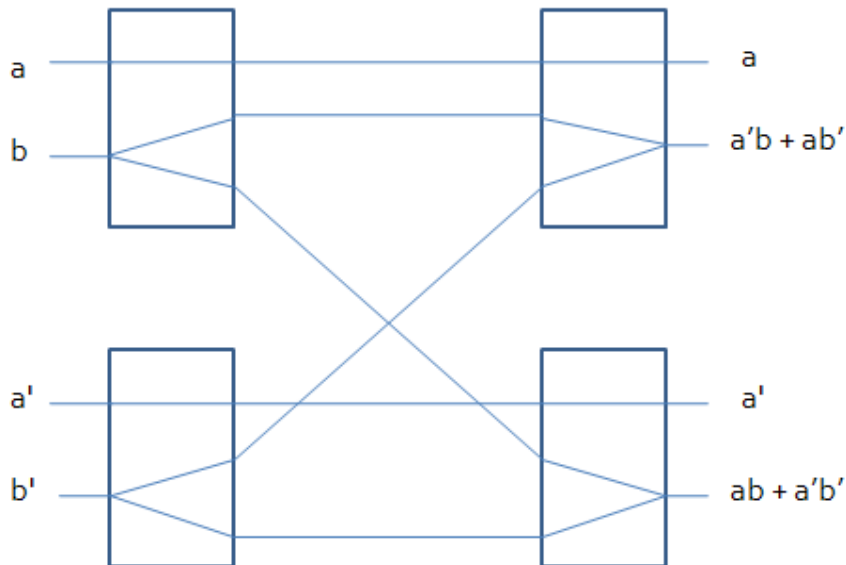


Figure 3.5.4: Double-rail Feynman gate implementation.

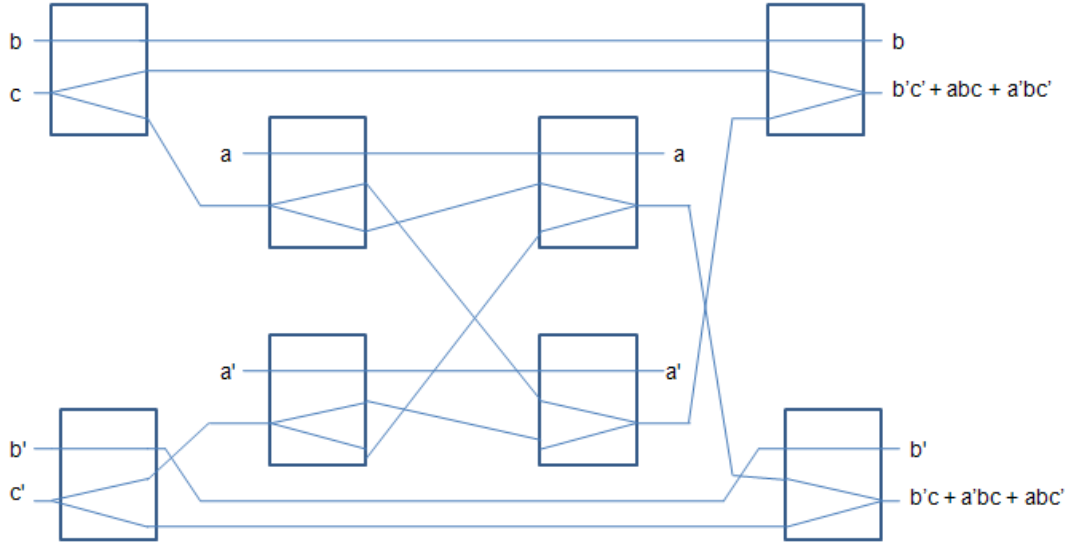


Figure 3.5.5: Double-rail Toffoli gate implementation

### 3.5.2. CONCLUSIONS ON Y-GATE SYNTHESIS

This section showed a simple solution to a new problem in logic synthesis: synthesis of reversible specifications of new type with Y-gates. The exhaustive search algorithm was created in Prolog. This approach can serve as an introduction to using Prolog in reversible logic synthesis. Because of its relational/reversible properties Prolog is ideally suited to reversible circuit synthesis. In the future, this algorithm can further be extended to large dual-rail reversible logic implementation with other gates. Technology-related simulations of synthesized circuits should be also performed to predict the power savings in them. All algorithms given in the dissertation can be easily implemented in Prolog as it is good for backtracking, search and pattern-matching rule selection.

Unfortunately Prolog is slow, so the next software will be written in C++. This is the area of future work.

## **Chapter 4: REVERSIBLE QUANTUM CASCADES AND THEIR SYNTHESIS**

### **4.1. Reversible Quantum Logic Cascades. Notations and analysis**

Quantum cascades are similar notations to reversible circuits from chapter 3. Compositional synthesis methods for such cascades have been presented for the first time in [Perkowski01e]. Many papers followed [Mishchenko02, Khlopotne02, Miller03a, Maslov03b, Padma03], which improved the original cascade synthesis algorithms introduced by the PSU team. The paper [Perkowski01e] did not formalize sufficiently some of the synthesis procedures, most importantly leaving the search strategy undecided. Subsequently new ideas were added by other authors. In this dissertation, I combine ideas from some of these papers with my original ideas and ideas from [Perkowski01e] to create new general models of new reversible gates, both binary and multi-valued, which generalize some of the well known binary quantum gates from previous chapters.

Observe that cascades are the simplest conceptually structures of circuits for the composition method. This is because they do not introduce new signals (ancilla bits). When a  $k \times k$  gate is applied for composition, it subtracts as many input variables as it has inputs and it adds the same number of new variables as its outputs. Thus the width of the cascade remains unchanged. Whether the cascade is built from inputs to outputs, or from outputs to inputs, it preserves the same width. Thus the synthesis algorithm does not have

to consider if more bits should be added during the synthesis process. Although some general algorithms in this dissertation do not assume cascades and they allow to create constant ancilla bits, these methods can be modified to create cascades with no ancilla bits. Sometimes it is good to allow having a small number of ancilla bits which shortens the cascade at the small added cost of increasing its width. The new ancilla bits are added in the middle of the cascade as it is being created. The number of these ancilla bits can be either decided by the user or selected adaptively by a smart tree searching algorithm that adds gates successively one by one. There is always a trade-off between the length and the width of the cascade, which should be taken into account by the synthesis algorithm.

The main differences between quantum and non-quantum cascades are the following:

1. SWAP gates are used in quantum cascades for wire crossing, technologies (T1-T3 discussed in chapter 3) do not require SWAP gates for wire crossing
2. Fan-out is not allowed in quantum cascades, some small fan-out inside gates is allowed in some technologies.
3. Ancilla bits are expensive in quantum technologies, not so expensive in quantum dots and other technologies.

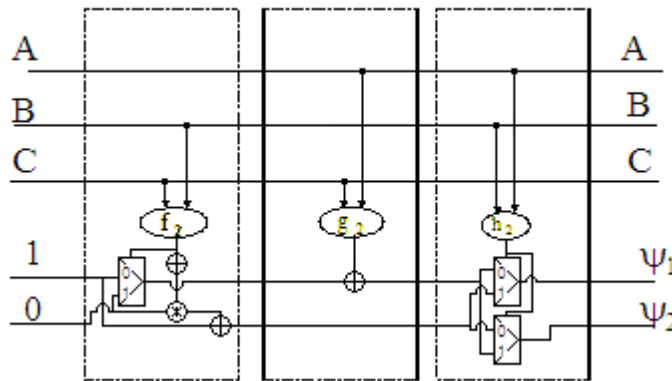


Figure 4.1.1. General cascade  
composed, from left to right, of

The circuit from Figure 4.1.1 is drawn using general notation of cascaded gates from chapter 3, but each of the gates can be redrawn (transformed) to the typical quantum notation. These types of transformations require sometimes adding ancilla bits internal to the gate.

Observe that inputs are forwarded to outputs (which is used in oracles) and that two ancilla bits (qubits, quantum bits in case of quantum circuits) are added at the bottom. Ancilla bit for  $\psi_1$  is initialized to 1 and ancilla bit for  $\psi_2$  is initialized to 0.

Some examples of cascades are shown in Fig. 4.1.1 – Fig. 4.1.13. where the upper part of the cascade, also called “control” part, (lines A, B, C in case of Fig 4.1.2a) always provides all primary inputs to control gates of the cascade, and the lower part, also called the “data path”, executes swapping and negation of signal (bottom two lines in Fig.

4.1.1). Many well-known standard logic synthesis methods can be adapted to synthesize reversible cascades of this type.

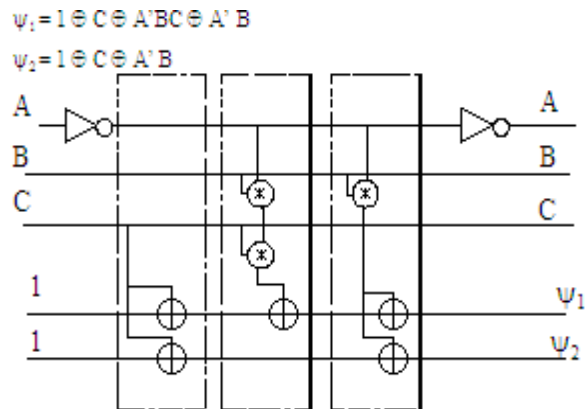


Figure 4.1.2a. Example of a multi-output Fixed Polarity Reed-Muller cascade from Toffoli Family of gates. (a) Reversible notation, (b) quantum notation has a more realistic gate model. Observe that each input variable has the same polarity in entire circuit.

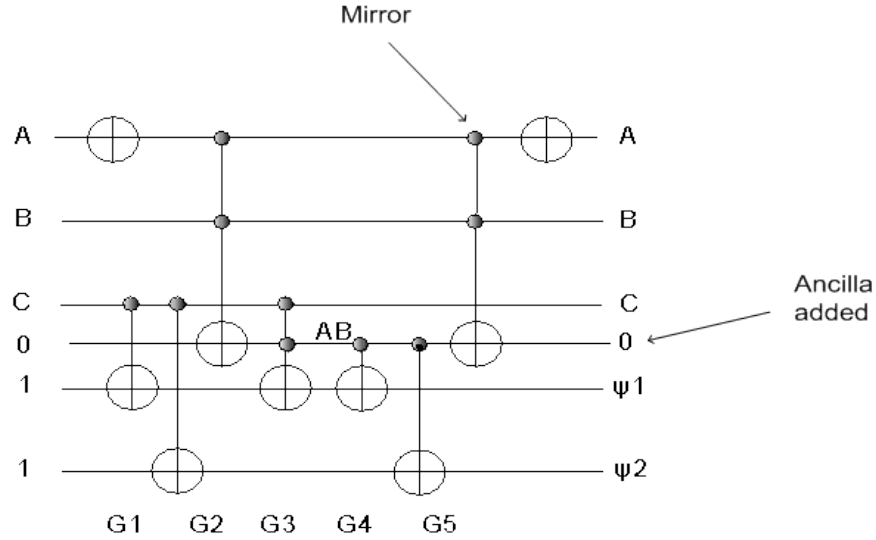


Figure. 4.1.2b. Yet another notation that shows more details of gates used. The quantum costs [Lee07] can be calculated for each gate from this cascade. This cascade is called the quantum array in many papers. Observe the role of the mirror gate that reintroduces constant value of the ancilla bit. This is used in quantum oracles.

For example, Fig. 4.1.2a shows a schematics of a Fixed Polarity Reed Muller form [AgrawalJha04b, Jha06] realized using generalized Toffoli gates. In this dissertation, we will use several notations: the classical logic notation and quantum array notation will be used most often, but we will sometimes, for convenience, mix notations or introduce new notations. Figure 4.1.2b presents the schematics from Figure 4.1.2a rewritten to the realization of quantum NMR technology. One ancilla bit was added, as well as a mirror gate. This form of presentation allows calculating quantum cost [Maslov05] of the whole circuit as the sum of the costs of individual gates in the circuit. Observe the reuse of

function  $AB$ . Observe that notations from Figure 4.1.2b and Figure 4.1.3 for circuit from Figure 4.1.2a are better to use when one wants to illustrate the realistic costs of gates that are used simultaneously in more than one outputs of the cascade. Small open circles denote control with negated input value (Figure 4.1.3). Observe repeated gates G1, G2 and repeated gates G4, G5. This is a standard notation for quantum arrays introduced by Richard Feynman.

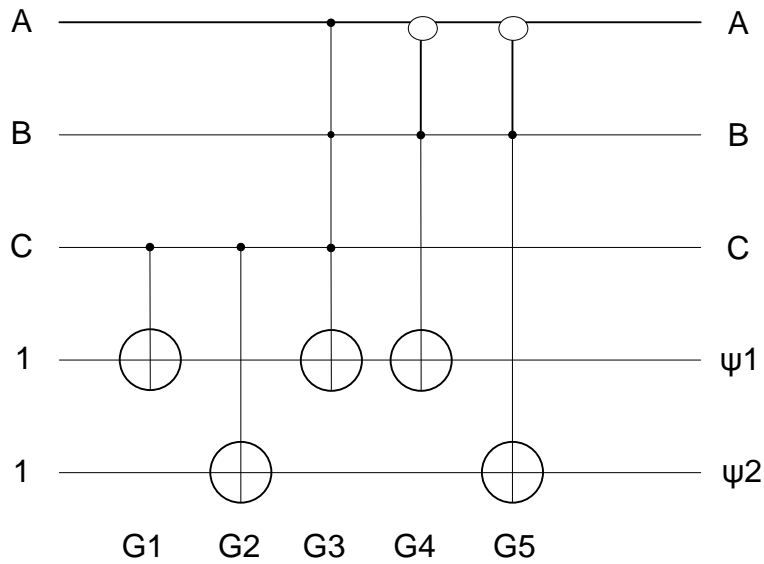


Figure 4.1.3. Example of a multi-output Fixed Polarity Reed-Muller cascade from Toffoli Family of gates. This is the same circuit as one from Figure 4.1.2 a, b but in yet another notation.

Figure 4.1.3 rewrites this circuit to the popular notation of binary quantum arrays invented by Richard Feynman. Notations shown here are used in journals on quantum computing, and emphasize either abstract schematics that is not directly realizable or a circuit from library gates that have definite cost and are realizable in a certain implementation technology.

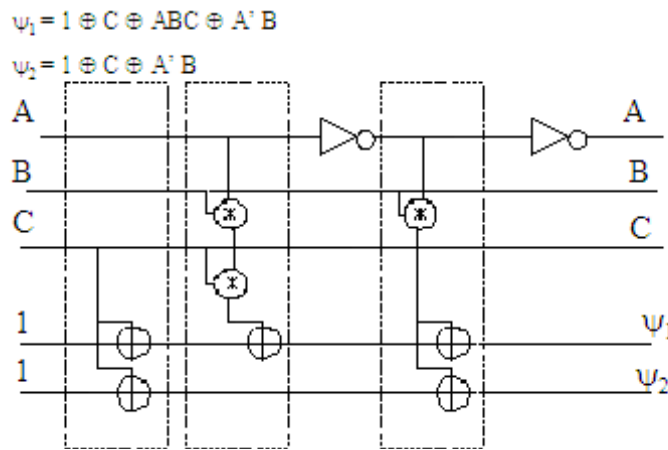
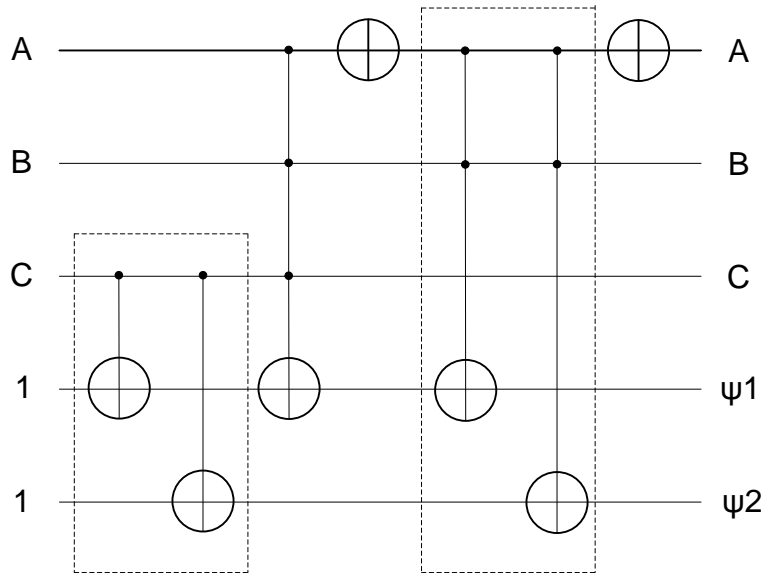


Figure 4.1.5. CMOS notation example of a multi-output ESOP (mixed polarity) from Toffoli Family of gates. Observe that variable A has different polarities in various Toffoli gates of the circuit.

Figure 4.1.5. presents an example of a cascade that realizes an Exclusive-OR Sum-of-Products (ESOP) circuits.



*Figure 4.1.6. Notation used in quantum circuits which emphasizes the repetition of gates.*

Figure 4.1.6 presents the same circuit in Feynman notation. Observe that all gates are controlled from black dots (positive polarities) and inverter schematics are used.

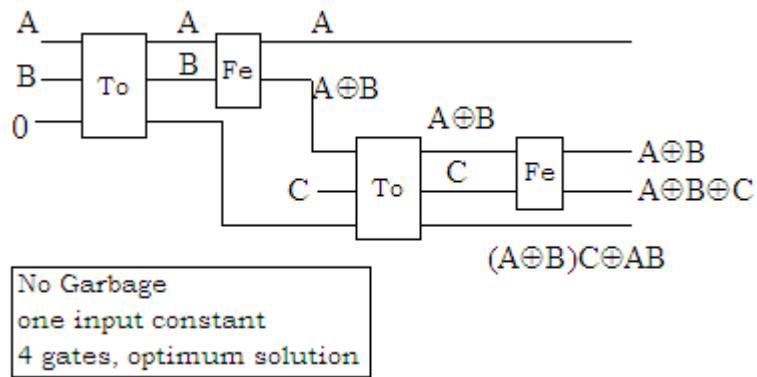


Figure 4.1.7. “CMOS circuit” notation for full adder realized using composition. Here we use  $To$  for 3\*3 Toffoli, and  $Fe$  for Feynman gates.

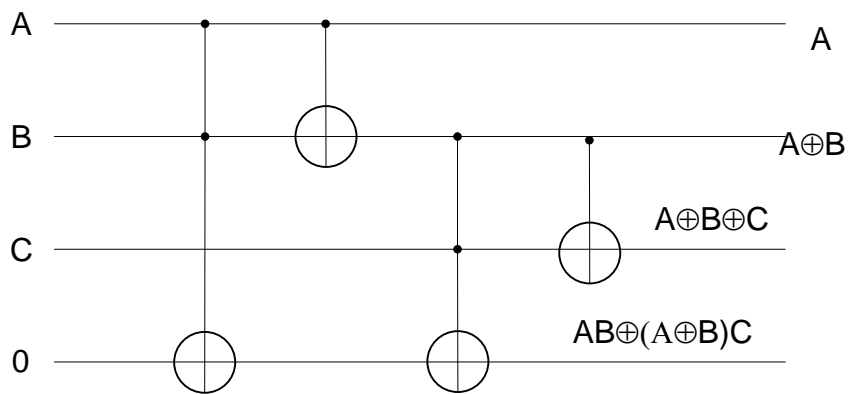
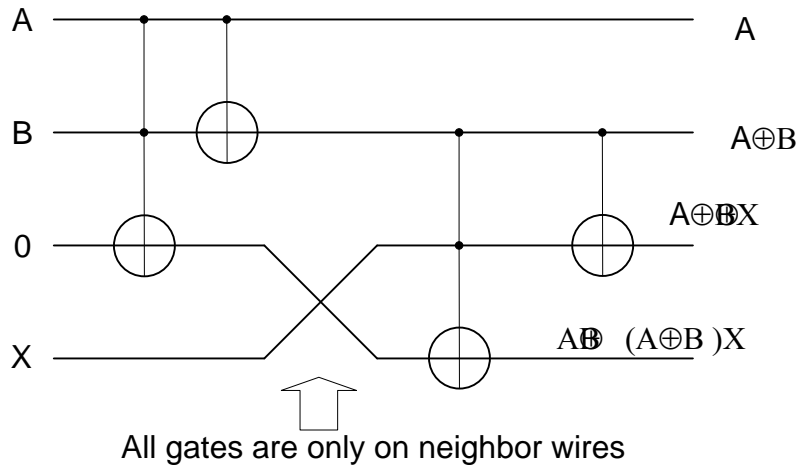


Figure 4.1.8. Example of a multi-output circuit – an adder. This is the circuit from Figure 4.1.7 in quantum array notation.



*Figure 4.1.9. An adder with gates on only neighbor wires. Such gates are used in Ion Trap Quantum Realization technology.*

Note that circuit from Figure 4.1.9 is the same adder from Figure 4.1.7 and Figure 4.1.8. In this notation, it is not allowed for a vertical wire to cross a horizontal qubit wire without a dot. It means that every gate is realized on neighbor qubits (linear array) only. A SWAP gate (second from left on qubits 0 and X) is realized using 3 Feynman gates.

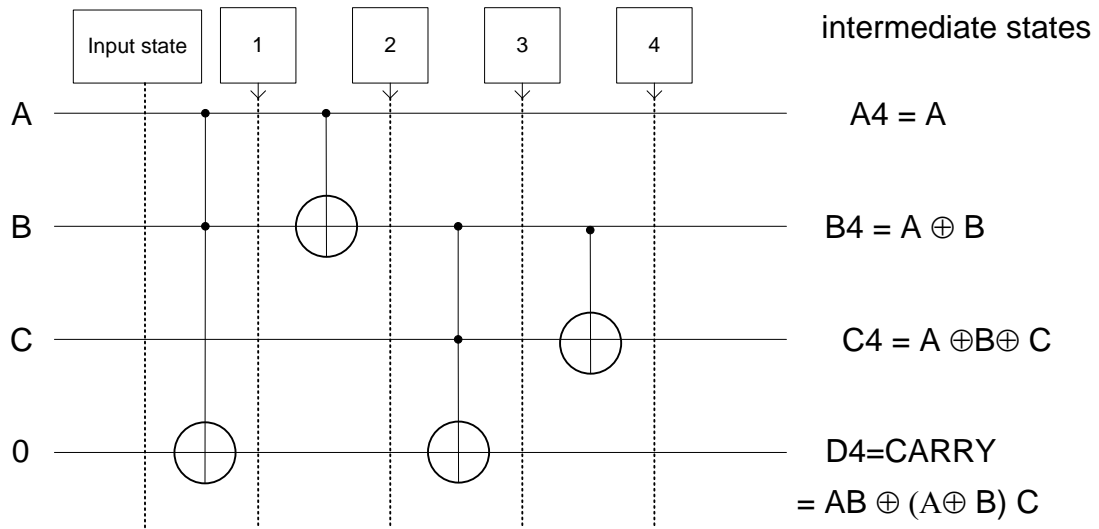


Figure 4.1.10. The adder from Figure 4.1.7 with vertical lines showing levels of reversible gates.

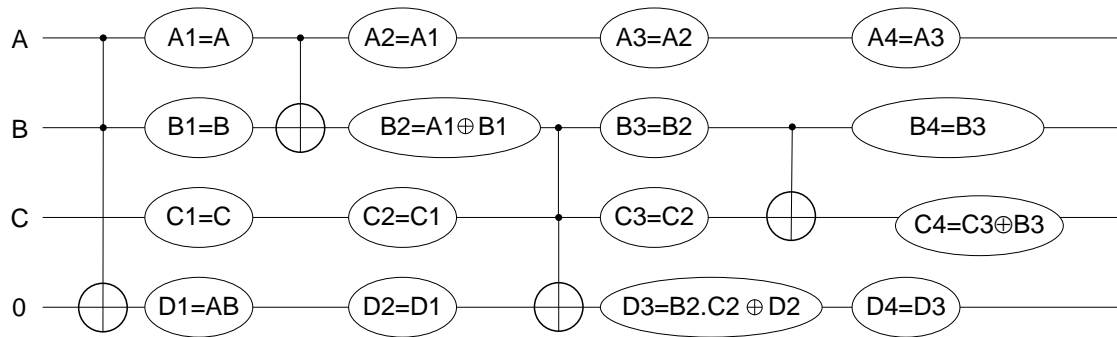
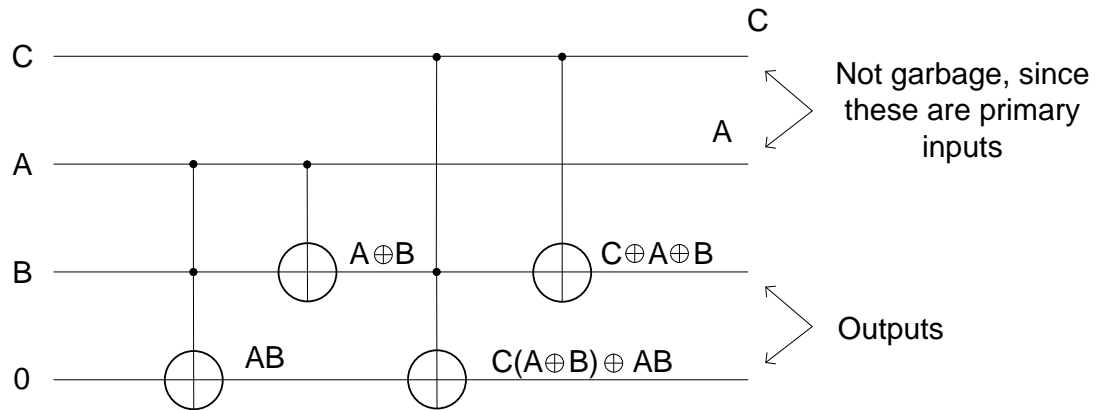


Figure 4.1.11. Adder from Figure 4.1.10 with intermediate signals illustrating steps of its analysis. Example of application of a level-by-level synthesis algorithm applied to our example.

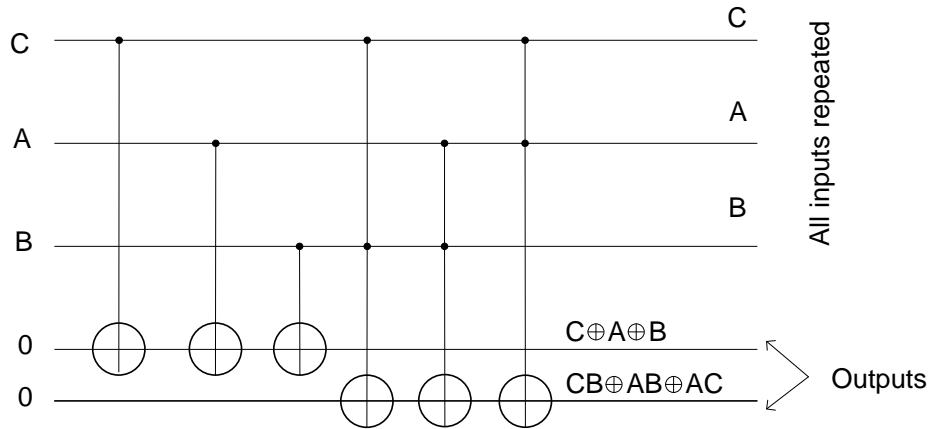
Figures 4.1.7 – 4.1.11 illustrate realization of a Full Adder using Toffoli (To) and Feynman (Fe) gates. The circuit is synthesized using methods introduced in this dissertation. The notation from Figure 4.1.7 is not concerned with the order (location) of inputs/outputs and using SWAP gates. It is used in adiabatic CMOS reversible technology. The quantum array of this circuit with order of wires A, B, C, 0 is given in Figure 4.1.8. In this circuit, wire C goes through the first gate. This can be avoided as in Figure 4.1.9 by changing order to A, B, 0, C and adding a SWAP gate. These transformations are important for linear Ion Trap technology in which the gates can be only on neighbor (geometrically adjacent) qubits. The same circuit is drawn in Figure 4.1.11 in “quantum array” logic notation to emphasize the generality of the method. The solution from Figures 4.1.7 – 4.1.11 and Figure 4.1.12 is the optimum solution. Let us discuss how it is created. It is first found that the original 3-input, 2-output function of the adder is not reversible and that it cannot be made reversible by adding one output signal (the reader can check this property using Kmaps from the definition of reversibility). Thus one more constant input is added and it is assumed that the width of the circuit (called the “scratchpad register width” in quantum logic) is four (see Figure 4.1.12).



## Width 4

Figure 4.1.12. Quantum array notation for Full Adder realized using

composition. The width of the “quantum register” is four. Note another order of inputs.



Width 5, six gates, two constant, not optimal but easy to find

Figure 4.1.13. Quantum array notation for Full Adder realized using PPRM type of logic.

This design style is used in quantum oracles in which all inputs must be repeated to outputs.

In the design from Fig. 4.1.12, we have two primary outputs, two potential garbage outputs, three primary inputs and one input constant. It was assumed in this design to not increase the width of the cascade and gates are selected to complete all primary functions and not generate garbages which would require mirror and spy circuits. Whenever a solution cannot be found given these assumptions and with the selected set of reversible gates, a backtrack is executed. Whether C and A are garbages or not, it depends on an application.

Observe that finding the structure of the cascade (with or without ancilla bits) for a given functional specification is a search problem. This search problem can be solved using the well-known search algorithms from Artificial Intelligence (AI) such as the breadth-first, depth-first, A\*, deepening, widening, best-bound or other search algorithms, even with no heuristics or with a very crude heuristics of choosing nodes of next levels. Thus even an algorithm with no heuristic cost function but based on only depth search limit of four would find ultimately the solution, like for instance one from Figures 4.1.7 – 4.1.9. Because the number of gates and wire permutations is high, such approach would be exhaustive and we need some heuristic cost function to guide it. First we used the Hamming Distance as the cost function, but it was not sufficient. Then we used a combination of Hamming Distance (will be defined next) and entropy (defined in a standard way) of EXOR of the wire function and the primary output function, for all pairs

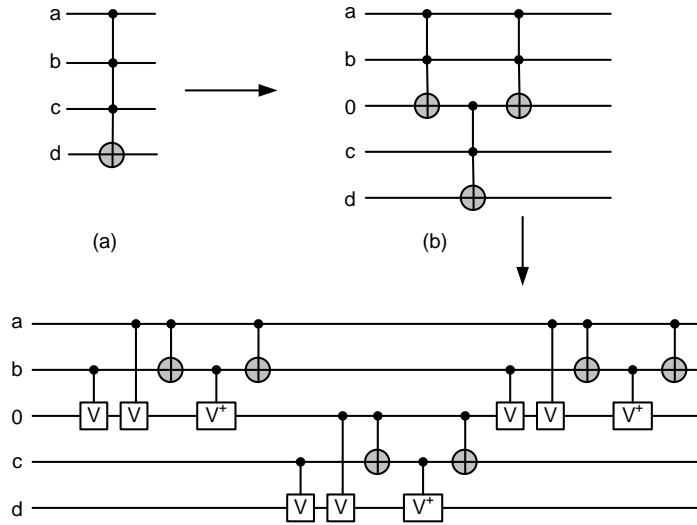
of wire and output functions. This was done for non-reversible initial function specifications. Observe that the entropy-based cost function has small values for Boolean functions with many zeros (false minterms) and many ones (true minterms) but has high values for functions with approximately the same number of ones and zeros. Functions such as an input variable or an EXOR of input variables have simple realizations but the highest possible entropy. This is the weakness of the entropy-based information theory cost functions. So, such functions were treated in a special way in our Greedy algorithm. Similarly all functions that are “products of literals” or “sum of literals” are treated specially in the cost functions. We continued to work on defining a better cost function for this task, as will be illustrated in Chapters 5 and 6. However, as we were not able to obtain convergence, we dropped this approach in favor of the algorithm presented in Chapter 6.

Let us illustrate now the synthesis using the adder example. After applying the first Toffoli gate from left in Figures 4.1.7 – 4.1.9, function AB is created which has high correlation with the primary output  $AB \oplus AC \oplus BC$ . Functions A, B, AB, C are sufficient to realize all primary outputs, so the next level of the cascade is now composed. Function  $A \oplus B$  is created as having a high correlation (a small value of the cost function) with respect to the primary output  $A \oplus B \oplus C$ . The variables after two input levels are now A,  $A \oplus B$ , C and AB. The Toffoli gate is selected which realizes directly the majority function. The variables are now A,  $A \oplus B$ , C and  $AB \oplus AC \oplus BC$ . Only one target output exists at this stage. It can be checked that Feynman gate is the best

choice since it realizes  $A \oplus B \oplus C$  and primary input C. Because previous levels created only function A as potential garbage, the function has no garbage because all other outputs than primary outputs are primary inputs – so that the energy put from power supply through inputs A, B, C will be returned to primary outputs A and C.

Another style of designing the adder is shown in Figure 4.1.13. Every output has its own ancilla bit and the inputs are replicated to outputs. Each output can be realized separately and is an xor of products of literals. We call it the ESOP realization style and it can use existing software for ESOP minimization. The results of using ESOP minimizer Exorcism-4 [Mishchenko03] were however not successful so we dropped this approach. It is still open for future research.

Another important topic is realization of large ( $k \times k$ ) Toffoli gates and how their costs are calculated. This is explained in Figure 4.1.14. This figure explains why the costs of large Toffoli gates grow quadratically. The whole circuit is decomposed to only  $2 \times 2$  truly quantum gates. Some of these gates such as Feynman are permutative. Other  $2 \times 2$  gates are Controlled-V and Controlled-V+ which are controlled gates that control truly quantum single-qubit gates called “Square root of NOT” and “Square-root-of-NOT-hermitian”, respectively. These gates are described by unitary matrices that are not permutative and are a base of many quantum realizations. Their details are not important to this dissertation and they serve only to explain the growth of costs and how to calculate the quantum costs.



*Figure 4.1.14. Synthesis of large Toffoli gate from 3\*3 Toffoli gates and next from 2\*2 quantum primitives such as Controlled-V and Controlled-V+. These primitives are not explained in the dissertation. The only goal of this figure is to show the approximate method to calculate quantum costs. The quantum cost is approximately the number of 2\*2 quantum primitive gates. This figure shows also the use of ancilla bit. (a) a 4\*4 Toffoli gate, (b) decomposition of a 4\*4 Toffoli gate to three 3\*3 Toffoli gates that uses one ancilla bit initialized to zero, (c) further decomposition of circuit from b to 2\*2 quantum primitives.*

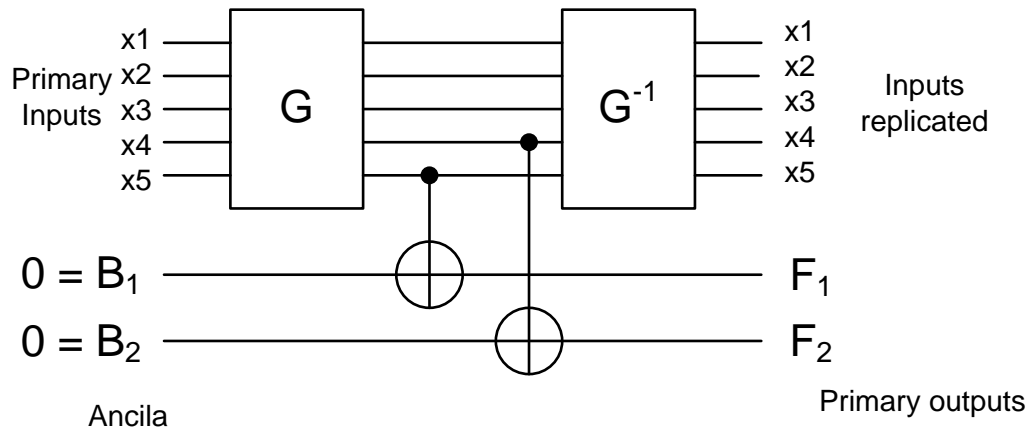
The notations, heuristics and principles outlined above are very useful to create all kinds of cascade synthesis algorithms, including exact, heuristic, evolutionary and MMD-like algorithms. The best heuristics so far are based on maximizing the number of identity mappings in reversible circuit described as a permutation (number of minterms that map to themselves). Next a combination of entropy and Hamming distance may be used. This dissertation discusses good choice of

criteria and orders of selecting the subfunctions (such as minterms) for the realization.

## 4.2. Background on Reversible Logic and Reversible Logic Synthesis

To avoid energy loss, the approach originated by Ed Fredkin and Tommaso Toffoli [Fredkin82] (and most subsequent early authors) has been next commonly used. It creates a “basic circuit”  $G$  from reversible gates with garbage outputs (Fig. 4.2.1). Next, this approach applies a “spy gate” for every primary output  $F_i$ . The spy gate is a Feynman gate with  $\mathbf{B}_i = \mathbf{0}$  which copies the output signal of the basic circuit. Next a mirror circuit  $G^{-1}$  is added with ancilla bit inputs being the second outputs of spy gates and the garbage outputs of the basic circuit. The mirror circuit is the inverse of the basic circuit and has as many gates (that are inverses to the basic circuit’s gates) as the basic circuit has gates. This solution leads to the duplication of the circuit’s delay and cost of gates. The delay is  $2n+1$  where  $n$  is the delay of basic circuit, and the gate cost is  $(3m + k)$  where  $m$  is the number of gates and  $k$  is the number of primary outputs (and spy gates). Therefore, all methods that do not take garbage bits into account while designing the basic circuit lead to very inefficient results. The main design requirement is that reducing the garbage in the basic circuit (ideally reducing to zero garbage, i.e. no garbage at all) is a good approximation for all quasi-optimal logic synthesis algorithms that use reversible gates. Let us

observe however that in some situations using the mirror circuits is the best solution or it cannot be avoided in cases when the primary inputs must be forwarded to the primary outputs. For example, one of two most famous quantum algorithms, the Grover algorithm, uses the so-called “permutative oracle” in which all primary inputs should be forwarded to the primary outputs and the decision function of the oracle is assigned to one ancilla bit. These repeated inputs are thus not treated as garbage bits. We are not going here to the depth of Grover algorithm which speeds up quadratically every NP problem, but we observe that it is this algorithm which motivates the research in permutative gates, circuits and blocks. The high usefulness of this algorithm is thus an important motivation of research performed in my dissertation.



*Fig. 4.2.1: The use of mirror circuit  $G^{-1}$  to an arbitrary reversible circuit  $G$  in a general-purpose reversible circuit with 2 outputs.*

The main differences of synthesizing a circuit with reversible gates, as compared to synthesizing a standard binary circuit, are the following:

8. In every reversible gate other than the inverter gate there are at least two outputs, while the synthesis methods of standard binary logic circuits assume one-output gates. It is easy to find reversible solutions from standard netlists in which every gate is replaced with its reversible counterpart that uses ancilla bits and garbage bits. As shown by Feynman this transformation is always possible to apply. Thus we sacrifice in each classical gate one or more output signals for garbage. Such solutions have not much practical value as they generate a very high number of garbage and ancilla bits. These methods are still used, however, because better methods are not yet known [AlRabadi01b, AlRabadi01a, AlRabadi02b, Mishchenko02, Perkowski01, Perkowski01a, Perkowski01b, Perkowski01c, Wille09].
9. A heavy price is paid for every garbage bit, whether the circuit just leaves the garbage bits unattended, or when the mirror circuit and spy gates are added, [Fredkin82], see also Figure 4.2.1. The garbage problem is strictly related to the ancilla bits problem, as we will see in next chapters. In quantum oracles all garbage bits

must be converted back to input variables (or constants) so the mirror circuits, in one form or another (global or local mirror circuits), must be used. Or, synthesis methods adapted from AND/EXOR logic are used, in which the gates are realized only on the function qubit. These last methods are however very inefficient for big oracles which include complex arithmetic operators.

10. In reversible logic, fan-out larger than one of any gate output signal is not allowed, every output can be used only once. Feynman gates can be used as “copying circuits” the same way as in the so-called “spy circuits” to increase the fan-out. However, for every fan-out of two a Feynman gate is used. Obviously, this approach also increases the cost and delay. (To use Feynman for copying is the assumption of most but not all “logic synthesis authors“, the “circuit design authors” allow for small violations by taking small fan-out  $> 1$ ).

Concluding, the main heuristic rules for efficient reversible logic synthesis of quantum cascades are the following:

1. Use as many outputs of every gate as possible as inputs to other useful gates. Thus do not create garbage outputs, that

either loose energy or require mirror circuits that lead in turn to loosing speed and increasing cost.

2. Do not create more constant inputs to gates (ancilla bits) than it is absolutely necessary. Some number of inputs constants are unavoidable, we should try to have a minimum number of gates with input constants. (For instance an input constant is unavoidable when the original function has three primary inputs and two outputs, function is not reversible, and it cannot be made reversible by adding one output. At least one constant input must be added to make it reversible).
3. Avoid leading output signals of gates to more than one input, because each such fan-out of two requires adding one copying circuit.

## **Chapter 5: SEARCH ALGORITHMS VERSUS SEQUENTIAL ALGORITHMS FOR BINARY REVERSIBLE CASCADE SYNTHESIS WITH NO ANCILLA BITS**

### **5.1. BRIEF PRESENTATION OF TOP ALGORITHMS FOR SYNTHESIS OF REVERSIBLE CIRCUITS**

Currently, there are two types of algorithms in literature to synthesize reversible circuits:

(T1) those like MMD [Agrawal04, Donald08, Dueck03, Gupta06, Kerntopf04, Khlopotine02, Maslov03, Maslov08, Maslov10, Miller03, Miller03a] that start from a reversible specification,

(T2) those that start from non-reversible specification and create ancilla bits like [Große09, Kumar07, Kumar08, Mishchenko01, Mishchenko02, Stedman04, Saeedi07, Saeedi07a, Saeedi07b, Wille08, Wille08a, Wille09].

The second type of methods has been successful for large functions [Große06, Große09, Wille08, Wille08a, Wille09] but these two groups of methods (T1 and T2) solve basically different types of problems.

The MMD algorithm (Miller, Maslov and Dueck) is currently the leading reversible logic synthesizer if no\_ancilla bits are used [Miller03a]. MMD uses permutation vector-like reversible function specification as its data. It generates no ancilla bits and uses no search. MMD software is reasonably fast and it distinguishes itself among other programs of this type as it achieves (theoretical) 100% convergence, regardless of the size of

problem being synthesized [Miller03a]. MMD can practically be applied to at most  $8 \times 8$  variable reversible functions and few larger functions of special types. This program is therefore the current benchmark for the evaluation of programs for reversible circuit synthesis. Due to MMD's non-minimal results, several research groups are constantly attempting to improve this algorithm since 2003.

Mathematically, MMD decomposes a large permutation of circuit's specification to small permutations of reversible gates. MMD uses the permutation vector-like reversible function specification as its input and internal data. Permutation vector corresponds to the truth table, so it is very large for functions with more than 13 variables. This large vector is explicitly used in the synthesis process and, thus, must be stored and processed in memory. Since it is intrinsically bound by the natural binary order of minterms, and hence does not use search, MMD cannot be enhanced through better search algorithms or iterative or recursive routines.

Since MMD processes only a single minterm order, this program is fast. Practically, however, very few reversible functions with more than 8 variables were presented as MMD benchmarks in the literature. It was found in our research, and by other researchers, that the complexity of both the synthesis process and the average circuit sizes synthesized by MMD grow very quickly with "large circuits" (above 8 qubits). In our research, it was difficult to evaluate the quality of our results for large circuits from reversible specifications chiefly due to the lack of a single solution for comparison.

Consequently, with this chapter, we set the benchmark for future research. Observe that recent papers on permutative quantum circuit synthesis use standard non-reversible specifications as their input, [Rice09, Wille09], while previous tools such as MMD only allow vector of permutations (only reversible) as the only form of function specification, highlighting the need to develop tools that convert irreversible specification to reversible. In any case, at this time MMD program is the current benchmark for the evaluation of programs for reversible circuit synthesis with no ancilla bits. A strong asset of the philosophy used in MMD, in contrast to those used in other programs is that MMD gives a warranty of convergence if the data is small enough for MMD to be able to keep them in memory. Due to the fact that the quality of MMD may be very low for functions where the exact minimal solution is known, several research groups are constantly attempting to improve on the MMD algorithm.

Agrawal and Jha's algorithm [AgrawalJha04b, Jha06] uses the number of terms in the Positive Polarity Reed-Muller (PPRM) expansion of synthesized functions as its cost function [AgrawalJha04b], and Kerntopf's algorithm uses the complexity of special BDDs, the SBDD's, as its cost function [Kerntopf04b]. The use of the cost functions based on complexities of ESOPs, FPRMs and other cascade types in the cost functions that guide the search have also been proposed in newer versions of composition-based search approaches [Khlopoutine02, Mishchenko02, Perkowski01e]. Although these algorithms are different than MMD in the sense that MMD uses no search, they belong to the same group of methods based on their assumption of using no ancilla bits. Kerntopf

used a new type of decision diagrams but did not prove convergence and, as a result, his method only worked for 3 variables. In unpublished research, we used ESOPs and FPRMs rather than PPRM but we were not able to find a heuristic that would work better than the variants from [Agrawal04, Donald08, Gupta06]. Other cascade types have also been proposed in newer versions of composition-based search approaches [Khlopoutine02, Mishchenko02] but there were troubles with either the size of solutions or convergence.

As PPRM (Positive Polarity Reed-Muller) expansion can be stored by an expression that is (on the average) shorter than  $2^n$  product terms, ( $n$  is the number of variables) the algorithms that use PPRM, could in theory minimize larger functions. On the other hand this algorithm has to store many PPRM equations as it represents a tree-search algorithm. Also, non-factorized PPRMs may be in many cases of similar complexity to truth tables, for instance for function  $f = a'b'c'd'$ . Some of the algorithm variants from [Agrawal04, Donald08, Gupta06] have trouble with convergence and there is a trade-off between provable convergence and size of circuits that can be minimized. A challenge thus still exists to create an algorithm that could trade-off quality for time, but with a provable convergence for every function. In this chapter we will present a new algorithm of this type.

After many failed attempts at creating better minimizers (in the sense of cost and size of function to be synthesized) based on other search strategies [Kumar08, Mishchenko02] and our own strategies not reported here, we decided to improve MMD. The main

weakness of MMD is that it is limited to functions of the size that their truth table (exponential size) can fit in memory. This limits practically MMD's approach to about 13 variables. Because of its design principle, even with big speed penalty MMD just cannot minimize larger functions. Thus an improved algorithm has to use an entirely different representation. When it was decided to use an internal representation other than a truth table or a spectrum with  $2^n$  minterms, the problem was “what is the best representation that would still guarantee convergence?”

Here we present a search algorithm MP that is both convergent, allows for synthesis of large functions, and produces near minimal solutions. This algorithm is based on various generalizations of MMD.

## **5.2. EXPLANATION OF THE MAIN IDEA OF MMD**

To make the dissertation completely self-contained, we give a brief overview of MMD. More can be found in [Dueck03, Maslov08, Maslov10, Miller03, Miller03a]. The main idea of all algorithms for reversible circuit synthesis of type T1 is to transform bit-by-bit a reversible function to its identity function.

### Example 5.2.1.

Fig. 5.2.1 illustrates the basic flow of MMD algorithm. The first column lists all input minterms of the function in the natural numerical order (linear): 0, 1, 2, 3, etc. The second column in Fig. 6.2.1 lists values of the output vectors that correspond to the input vectors from the first column. For example, the input minterm  $a' b' c' = 000$  is mapped to the output minterm  $A' B' C' = 000$  and input 001 is mapped to the output minterm 100. Self-mapping minterms are minterms with matching input and output values (e.g., minterm 000 above). MMD applies successive gates to the output column (ABC), bit-by-bit, to generate the corresponding minterm of the input column (abc). Recall that Toffoli (Feynman) gates are used that are self-inverse gates ( $M^{-1} = M$ ), so they process information the same way from inputs to outputs and from outputs to inputs. The MMD algorithm shown here is thus a “backward searching” algorithm or “output to input searching” algorithm. Since the first minterm is self-mapping, MMD skips to the second minterm applying a controlled-Feynman gate to *bit c*, shaded, conditional on *bit a* being set, underscored. After the application of each gate, the output column of minterms (of intermediate functions) become more and more similar to the first column – the column of input vectors. The question is “*what does it mean to be more similar?*” It is an advantage of general search methods that various measures of complexity or coincidence or similarity have been used [Kerntopf04, Khlopotine02, Mishchenko02]. This may lead to better and faster solutions but it is hard or impossible to prove convergence. The MMD

algorithm has however a very simple and working solution to this problem. It requires that intermediate columns to remain exactly the same as the input column in some subset of rows from the top. The *completed rows*, start from row 0, then row 1, row 2 etc. up to the minterm under construction. When some subset of rows from top are completed, they are not allowed to be changed again (shown in shaded areas in Fig. 5.2.1) which is guaranteed by the selection of proper control bits.

abc	AB	1	2	3	4	5	6
	C						
000	000	000	000	000	000	000	000
001	100	<u>101</u>	<u>001</u>	001	001	001	001
010	101	<u>100</u>	100	<u>110</u>	<u>010</u>	010	010
011	001	001	<u>101</u>	<u>111</u>	<u>011</u>	011	011
100	110	<u>111</u>	<u>011</u>	011	<u>111</u>	<u>101</u>	<u>100</u>
101	010	010	010	010	<u>110</u>	<u>100</u>	<u>101</u>
110	011	011	<u>111</u>	<u>101</u>	101	<u>111</u>	<u>110</u>
111	111	<u>110</u>	110	<u>100</u>	100	<u>110</u>	<u>111</u>
		$a \rightarrow c$	$c \rightarrow a$	$a \rightarrow b$	$b \rightarrow a$	$a \rightarrow b$	$a \rightarrow c$

Figure 5.2.1. MMD method illustrated with truth tables of intermediate functions. Notation  $a \rightarrow c$  means  $c = c \oplus a$  means “flip  $c$  if  $a=1$ ”. Control lines are underlined and affected bits are shaded.

This is the main idea of the MMD algorithm and actually, the only new algorithmic idea of this method (excluding templates). The proof that this algorithm is convergent is obvious as every step creates one more bit in a row from top that is the same in the intermediate column as in the first column. This way, after at most  $n * 2^n - 1$  steps (intermediate columns) the last column becomes exactly the same as the first column, and thus, the remaining function to be realized is an identity function (a better bound was also proven by Maslov but it is not relevant here). As we see, the strength of this algorithm is the guarantee of convergence, but since the complexity is exponential (in terms of time and space), MMD is limited in application to a small number of bits. So far, however, MMD continues to represent the benchmark to meet as no better algorithm had been proposed. The symbol  $a \rightarrow c$  in the column 1 means that whenever  $a = 1$  in the previous column, the *bit*  $c$  is flipped from 0 to 1 and from 1 to 0. Hence, this transition from column to column executes the Toffoli gate  $c = c \oplus a$ . The reader may check that the number of completed rows is either the same or larger from column to column. In this example the upper complexity bound is  $n * 2^n - 1$  which for our 3-bit example yields  $(3 * 2^3 - 1) = 23$  gates. Note that in this example, MMD created a circuit with only 6 gates. Here MMD happened to work well. But there are examples where the gate number is close to the upper bound although the minimal number of gates is lower.

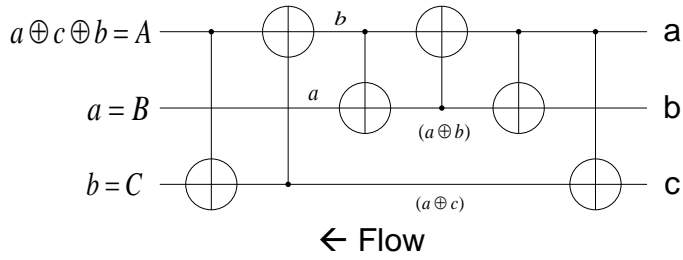


Figure 5.2.2. The solution circuit found from MMD in Fig. 5.2.1 drawn and created from outputs to inputs. The arrow shows the flow of signal from inputs to outputs. This method is possible because each reversible gate used in this figure is its own self-inverse.

### Example 5.2.2.

This example (Figure 5.2.3a) illustrates MMD graphically as a tree search, of which only one branch is shown, growing from top to bottom. This branch corresponds to the succession of columns as used in Figure 5.2.1. Instead of columns of binary vectors as in the previous example, new intermediate functions are represented as K-Maps on primary input variables. As we see, we follow the MMD principles by selecting the minimal non-completed minterm in binary order. This is the same principle as in the previous example, but for explanation we use a different representation. The solution is shown in Figure 5.2.3b.

In Figure 5.2.3a, the first map on top realizes the Fredkin gate. The first selected gate (from the left) in Fig. 5.2.3. is Toffoli described by equation  $c = c \oplus ab$ . This creates the map drawn below this operator, the second map from top. The next gate, Toffoli  $b = b \oplus ac$  is selected which creates intermediate functions  $A^2(a,b,c)$ ,  $B^2(a,b,c)$ ,  $C^2(a,b,c)$ .

Selection of gate  $c = c \oplus ab$  creates identity K-Map, and thus completes the search. The shaded cells are “completed” in the sense of MMD.

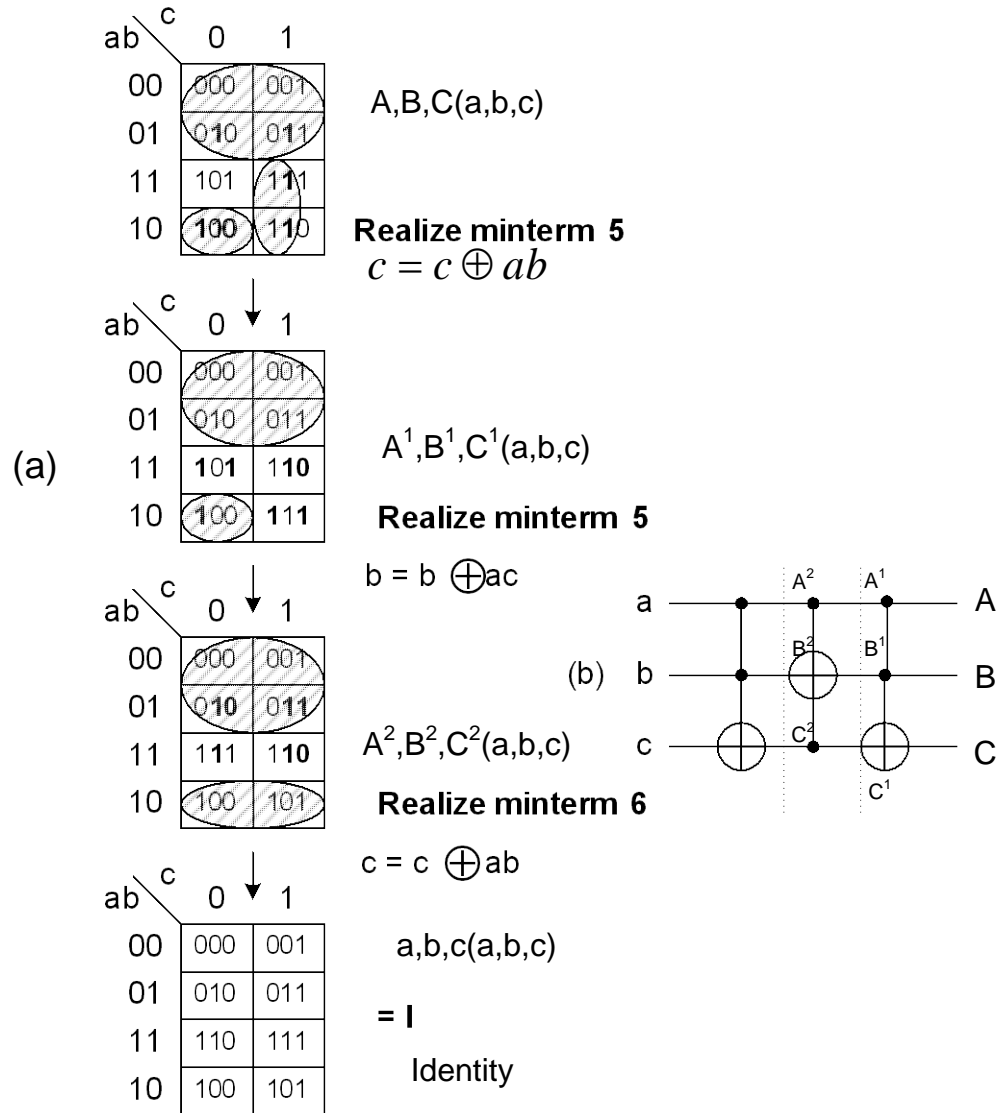


Figure 5.2.3. (a) Graphical visualization of MMD algorithm. A Branch of a search tree is shown with the final function as the initial node (K-Map). Arrows correspond to applying operators which in MMD are Toffoli and Feynman gates (only Toffoli here). (b) The final solution, non-

minimal. The intermediate signals  $A^1, B^1, C^1$  are created after applying Toffoli gate to primary outputs  $A, B, C$ . Similarly after applying the second Toffoli gate, the intermediate signals  $A^2, B^2, C^2$  are created. The next signals created are  $A^3 = a, B^3 = b, C^3 = c$ , so the search is terminated as identity has been found. The first map in (a) shows the situation when we flip bit  $c$  in these cells in which bits  $ab = 11$ . The second map is when we flip bit  $b$  in cells in which  $a = 1$  and  $c = 1$  (these bits are shown bold). The third map shows the synthesis stage when we flip bit  $c$  in these cells in which bits  $a = 1, b = 1$ . The final map at the bottom is when reaching identity completes the search. Every branch of our search tree terminates with a leaf that is either an identity or a branch is interrupted because the search in this branch is evaluated as not prospective and not leading to a potentially better solution.

(a)	ab \ c	0	1
	00	0	1
	01	1	0
	11	0	1
	10	1	0

(b)	ab \ c	0	1
	00	0	0
	01	0	0
	11	1	1
	10	1	1

(c)	ab \ c	0	1
	00	0	0
	01	1	1
	11	1	1
	10	0	0

A

B

C

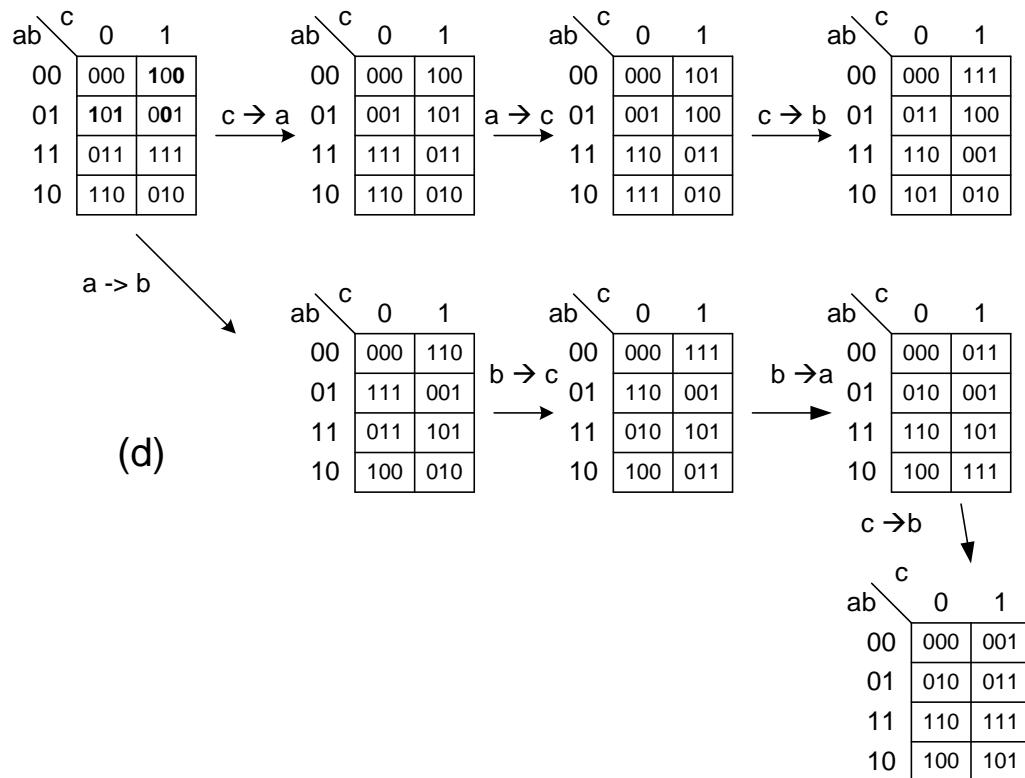
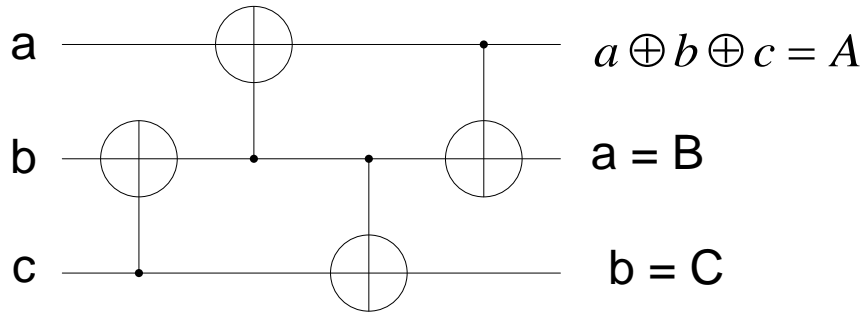


Fig. 5.2.4. Another search for function from example 5.4.2.1. (a) Function  $A(a,b,c)$ , (b) Function  $B(a,b,c)$ , (c) Function  $C(a,b,c)$ , (d) Search tree with two branches. The upper branch is not completed. The lower branch leads to the improved solution from Figure 5.2.5.

### Example 5.2.3

Figure 5.2.4 shows graphic tree search for function from Fig. 5.1.1, example 5.2.1. Observe that a solution from Fig. 5.2.5 is found, which has four CNOT gates, while the quantum array in Fig. 5.2.2 had 6 CNOT gates. We obtained a better solution thanks to violating the ordering from the MMD algorithm. This illustrates that it is reasonable to find for new orderings as they may improve the solution costs.



*Figure 5.2.5. The optimal circuit found by modifying the search in MMD. It is created in the lower branch of the tree from Figure 5.2.4, gates created from outputs to inputs.*

### Concluding on complexities and qualities of algorithms.

1. MMD misses best solutions. Adding search and search heuristics to MMD can help, especially by modifying the selection order for minterms.
2. MP adds search and heuristics to MMD thus improving costs of solutions.

3. MP introduces new representation, thus decreasing dramatically the space complexity.
  - a. Time complexity of MMD (without template matching) is of the order of  $n * 2^n - 1$  steps for a function of  $n$  bits and it does not depend much on the function.
  - b. Space complexity of MMD is of the order of  $n * 2^n - 1$  steps for a function of  $n$  bits and it does not depend on the function.
  - c. Our algorithm MP has time complexity  $O(k * n * 2^n)$  where  $k$  is a constant, and has space complexity  $O(nP(n))$ , where  $P(n)$  is some polynomial of  $n$ , related to the complexity of expressions describing the function. Therefore MP trades-off the increased synthesis time for improved size of functions and quality of solutions.
  - d. It is well-known that the expressions of those Boolean functions that are practical examples are much less complex than the worst-case specifications of these functions represented by exponential forms such as truth tables. For instance function of 6 arguments  $F(X1, X2, X3, X4, X5, X6) = X1 \oplus X2 \oplus X3 \oplus X4 \oplus X5 * X6$  has the “expression complexity” of 6 (number of literals), and the “truth table complexity” of  $2^6 = 64$ .
4. One may argue that in the worst case the expression complexity is also exponential, so why to use expression complexity as a complexity measure rather than the truth table complexity. This question is related to the well-known discussion in CAD community in early 1980's. Some, mathematically-minded,

researchers argued for sampling of statistically generated Boolean functions as an “objective” measure of algorithms’ quality. They argued also and proved mathematically that some logic synthesis methods such as Ashenhurst-Curtis Decomposition are in their mind “not practical” as “statistically all functions are not decomposable”. On the other hand every digital design engineer knows that Boolean functions such as adder, multiplier, state machines and most if not all “real life” circuits are decomposable. It was next shown that mathematically “nearly all” Boolean functions have “bad” properties and cannot be decomposed or represented by expressions of less than exponential complexity. Hopefully, CAD software is in most cases used for synthesis of “industrial” functions. Such functions specified in VHDL or Verilog have some internal logic related to their intended logical or arithmetical application and are thus not “worst case” functions. Hence came the idea supported by IBM, Bell-Labs, GE, other industrial companies and University of California Berkeley to create sets of industrial benchmarks and to test algorithms on these benchmarks. In some rare cases like cryptography circuits the complexity of expressions is nearly exponential and industrial benchmarks are not a good test of algorithms’ quality, but in most cases CAD community evaluates new software tools based on “industrial benchmarks” such as MCNC or ISCAS, rather than on worst case statistical samples. This is the methodology that we took in this dissertation.

### 5.3. What is wrong with MMD?

Last example shows that MMD can create non-optimal solutions even for very small functions. Figure 5.3.1 is an example. Here the minimal Fredkin gate is found in one branch of the search tree. The KMap illustrates the original function on original variables and next each node of the tree illustrates an intermediate function (with respect to the original variables) until the identity map is found, which terminates the algorithm. The same method is applied to create new maps. If the MMD ordering is used, the solution would have been be more expensive. The ordering in Figure 5.3.1 violates MMD but finds the best solution. The shaded area, represents the “completed minterms”. The bolded binary symbols 0 and 1 correspond to selected values to be changed. Let us look at the first node of the tree. The KMap in this node has all the completed minterms shaded with interrupted oblique lines. Minterms (cells) 0, 1, 2, 3, 4 are shaded as they were completed. The smallest minterm that is not shaded has number 5. We want to change its smallest (right) bit from 0 to 1. I select gate “Feynman” controlled by bit b. Observe that all bits are bolded in this KMap. Also note that this selection changes the contents of cell 010. This selection would be therefore not allowed in MMD.

The next KMap below corresponds to the result of applying Feynman gate  $c = c \oplus b$ . In this new map, there are 5 cells that are not completed (Fig. 5.3.1). Again, the number of such cells increased, which is in contrast to MMD where the number of completed cells always grows, thus the number of cells that are not completed decreases. The non-

completed minterms are: 010, 011, 110, 111 and 101. Again in this situation MMD would select cell 010 with contents 011. But we select cell 101 with contents 111. The bold symbols are those to be changed by gate selection. These are binary symbols corresponding to variable b. So I apply gate “change b when  $ac = 1$ ” which is  $b = b \oplus ac$  or Toffoli gate with EXOR in the middle bit. Thus the next gate is created which has 4 cells that are not self-mapped, they are 010, 011, 110, and 111. We see in this map that bit c is not in order so we have to flip this bit. The map tells us to do this when  $b=1$ . This leads to the lowest KMap at the bottom of the figure. In this map every cell is mapping to itself, including minterm 110 mapping to 110 and minterm 111 mapping to 111. This map is then identity, so our algorithm terminated. We see that we violated MMD rules several times, but were still able to find the exact minimum solution.

The graphical visualization method used here was developed to help understand MMD and also to create new tree search methods to minimize reversible functions that hopefully would lead at the end to an optimal or at least an improved algorithm. The visualization helps creating and analyzing such various search strategies.

Concluding that using other ordering than MMD, can lead to a better solution (in terms of cost). The question is: “*What are all good orderings to be used as branches of the search tree?*”

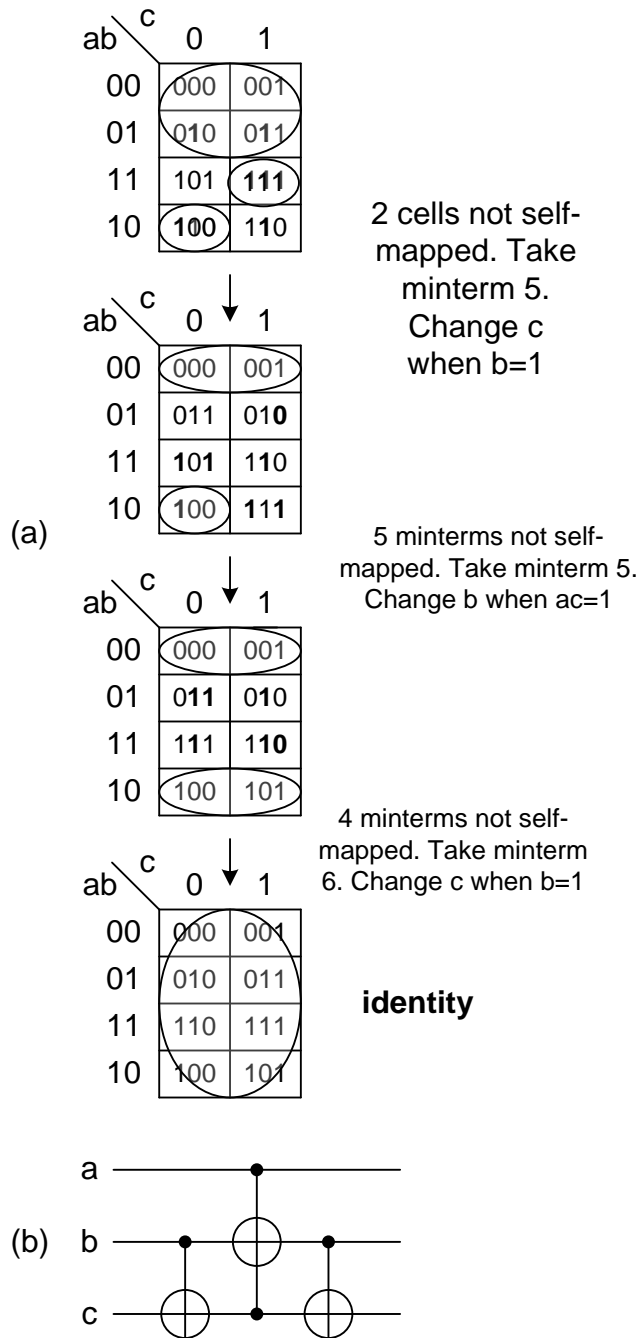


Figure 5.3.1: Design of Fredkin gate. (a) Optimal order shown in the branch of the search tree above violates the natural order used in MMD (MMD order), (b) the solution circuit found in the process from Fig. 5.3.1a which is the well-known minimal circuit of Fredkin gate.

#### 5.4. MMDS AND MMDSN ORDERINGS

The main concept of MMD's natural binary minterm ordering was challenged in [Stedman04] as the only 100% convergent ordering. It was found that MMD's minterm ordering falls into a subset of orderings that do not exhibit certain important property that was called the "control line blocking". This observation lead to the creation of the "MMDS ordering" [Stedman04]. To make this chapter self-contained, all these ideas will be defined below. Without any backtracking, bi-directional search or any template matching, the MMDS ordering was used exhaustively (using all possible orderings) and was superior (in term of cost of circuits synthesized) for 3-bit circuits [Stedman04]. The MMDS orderings can be used with any number of input variables gaining an advantage over MMD when the number of inputs increases. However, the number of MMDS orderings is too high to use them all for synthesis. In this chapter, we introduce a subset of the MMDS orderings, herein MMDSN orderings, which greatly reduces the number of terms examined while providing near minimal cost solution superior to MMD.

MMD stipulates that the function is arranged in a natural binary code order by input assignments. Each iteration adds a gate in order to correctly transform the output minterms to match the input minterms without changing any of the previously *completed* (from top rows) output minterms. Other innovative algorithms utilized a greedy

algorithm where gates are chosen to reduce the cost function from input to output. For example, Hamming Distance determines the choice of gates to transform the output function to the original function or to the identity function. Such algorithms did not always converge, unlike, MMD which guarantees convergence but might give the worst solutions (in term of cost). So, how can we combine these two main ideas of natural ordered search of MMD and greedy search to improve the quality of results and always achieve convergence? Such combination is a goal of our research.

A good ordering should not conflict with the MMD's main idea of not changing any previously set outputs. This idea is also what guarantees MMD's convergence.

**Definition 5.4.1.**

The condition of *Control Line Blocking* occurs when all control lines of the current minterm are a subset of the control lines of a previously completed minterm in the input order, making it impossible to change any output bits during the current iteration for previously completed minterms.

*Mathematical Check =>*

*if #later = #later & #earlier*

*then there is control line blocking*

Example 5.4.1. Control line blocking exists

$$101 = 101 \ \& \ 111$$

Example 5.4.2. Control line blocking does not exist

$$001 = 101 \ \& \ 011$$

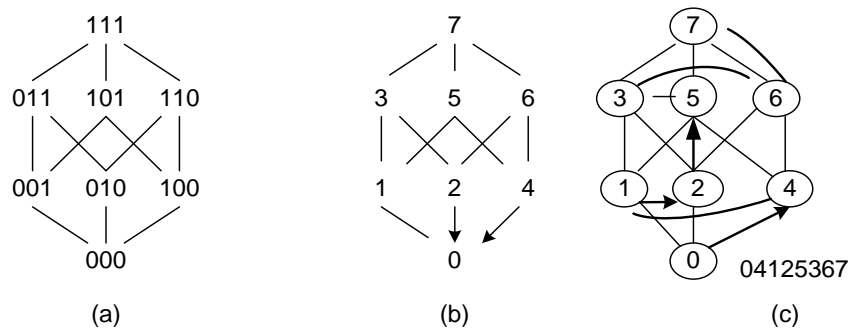


Figure 5.4.1. New orders for MMD-like synthesis. (a) Hasse diagram with binary vectors, (b) Hasse diagram with natural numbers, (c) Ordering of nodes that violates the MMD order, illustrated on the Hasse Diagram. This is however a valid MMDS ordering.

Therefore, any ordering of inputs that does not lead to the occurrence of the blocking condition can be used in an improved MMD algorithm. The method to find all non-blocking permutations for any number of inputs was found in [Stedman04]. No control line blocking seems to be a very restrictive rule. For a three-input function, there are initially 8! (40,320) permutations. Therefore there are the same number of various orderings. Instantly that number is reduced to 6! Since 000 must come first and 111 must come last. Using the software, 48 permutations, called MMDS orders, were found to

exhibit no control line blocking for all 3\*3 reversible functions. Included in this set is the original MMD ordering.

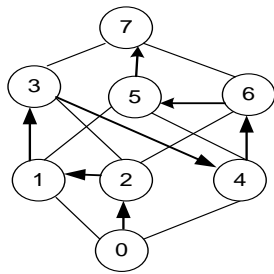
The binary vectors of cells (minterms) of a 3 \* 3 reversible function can be represented as a well-known Hasse Diagram, where a bit-by-bit domination relation ( $1 \geq 0, 1 \geq 1, 0 \geq 0$ ) is used as an ordering relation (see Fig. 5.4.1a, b). While binary vectors are used in Fig. 5.4.1a, the Fig. 5.4.1b uses natural numbers being counterparts of these binary vectors. The rule says “*never to take a dominating node (number) before a dominated node*”. Thus 5 cannot be taken before 1. As we see, MMD order satisfies these rules. Other good orders are shown in Figs. 5.4.1c and 5.4.2.

As the number of input lines increase, the number of non-blocking orderings increase exponentially. For functions with four inputs, there exist 1,680,382 non-blocking orderings. As the amount of non-blocking orders increase, the optimality of the results from MMDS increase as well, and as a result, synthesis time increase. With MMDSN orderings, a set of rules were created to distill the best possible control choices from the set of all possible control line choices, as follows:

- The target bit cannot be used to control the current transformation,
- Use minimal number of control bits necessary to flip the target bit,
- Cannot change previously completed (transformed) minterms.

- Process  $0 \rightarrow 1$  transitions first to maximize availability of control lines, and hence, guarantee convergence.

The choice of control lines is then sent to the gate choice function to produce a circuit. Control line blocking is the only rule that guarantees convergence, a subset of all non-blocking input orderings can easily be found, and it can be easily proven that all non-blocking input orders will converge for all output permutations.



*Figure 5.4.2. New ordering 02134657 for MMD-like binary synthesis, a valid MMDS order which is consistent with the Hasse diagram relations of orderings.*

**Theorem 5.4.1.**

*All non-blocking input orders converge for all output permutations.*

Proof of Convergence:

Convergence is guaranteed in MMD and MMDS by guaranteeing that no previously transformed minterm will be affected by future transformations (non-blocking of control lines). All following output bits are able to be changed without altering any previously set outputs. This is guaranteed because the input orders do not exhibit control line blocking. With MMD and MMDS' methodical approaches, as long as all output bits can be

changed without altering any previously set outputs these algorithms will converge every time.

MMDS set of orders is a superset of MMD. Our improved algorithm uses multiple MMDS input orders that exhibit no control line blocking. Included in these orders is the MMD natural binary order. MMDS ordering algorithm performs the same bit transformations strategy for all non-blocking input orders, and reduces the circuit more than the standard MMD algorithm. This outcome is obvious, given that MMD is a subset of MMDS, so it can perform no worse than MMD.

**Definition 5.4.2**

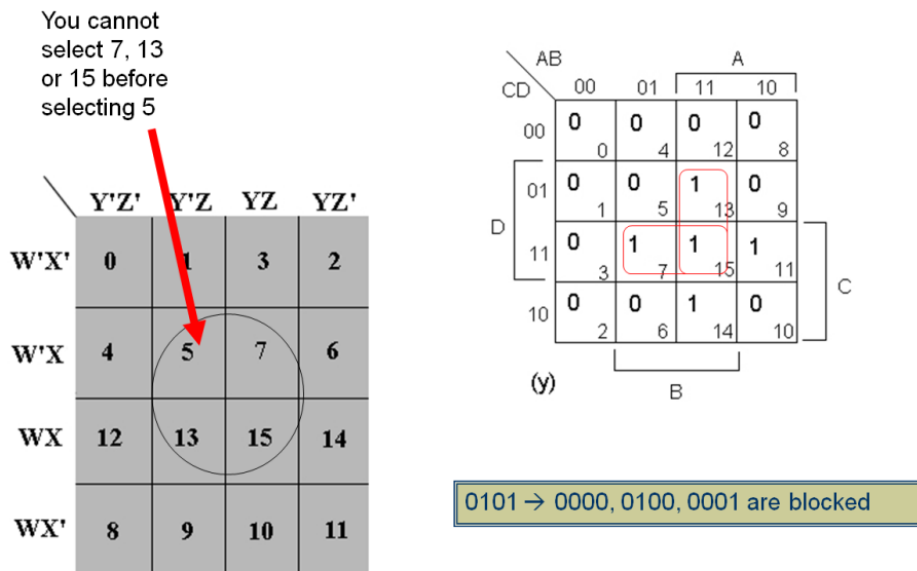
*MMDSN order is one in which the minterm  $00...0$  is generated first, followed by all minterms with a single one(1) in random order, followed by all minterms with two ones (1's) in random order, and so on, successively incrementing the number of ones (1's) in each band until we finally reach the minterm  $11...1$ .*

Example for 3 variables: MMDSN order is for instance: 000, 100, 010, 001, 110, 101, 011, 111. This is also an MMDS order but not the MMD order.

Figures 5.4.3 – 5.4.5 explain visually, using K-Maps the constraints imposed on correct (MMDS) orderings. These KMaps are useful to generate all MMDS orders or MMDSN orders for graphical minimization methods. These maps do not introduce anything new to

the theory, I added them however here for the sake of explanation of the crucial issues of ordering and their visualization. For me such maps were very useful to understand the problems and create new algorithms, so I believe they will be also useful for the reader.

1



2

Figure 5.4.3. Graphical visualization of MMDS orderings in Karnaugh Maps and explanation of the concept of blocking, also the rule and enumeration of cells of the KMap.

Allowed order for MMD

	$Y'Z'$	$Y'Z$	$YZ$	$YZ'$
$W'X'$	0	1	3	2
$W'X$	4	5	7	6
$WX$	12	13	15	14
$WX'$	8	9	10	11

3

Figure 5.4.4. Graphical visualization of MMDS orderings in Karnaugh Maps. Beginnings of the orderings.

Allowed order for MMDS

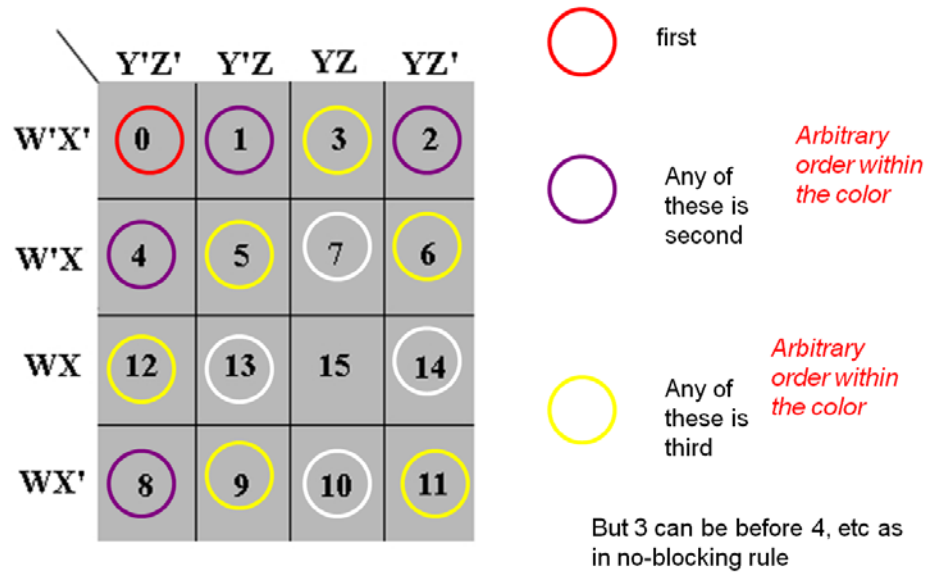


Figure 5.4.5. Graphical visualization of MMDS orderings in Karnaugh Maps. Orderings visualized with colors.

Part of the tree search developed by the tree search method to generate all 48 MMDS orderings for functions of 3 variables is shown in Figure 5.4.6 with binary strings replaced with their natural number counterparts.

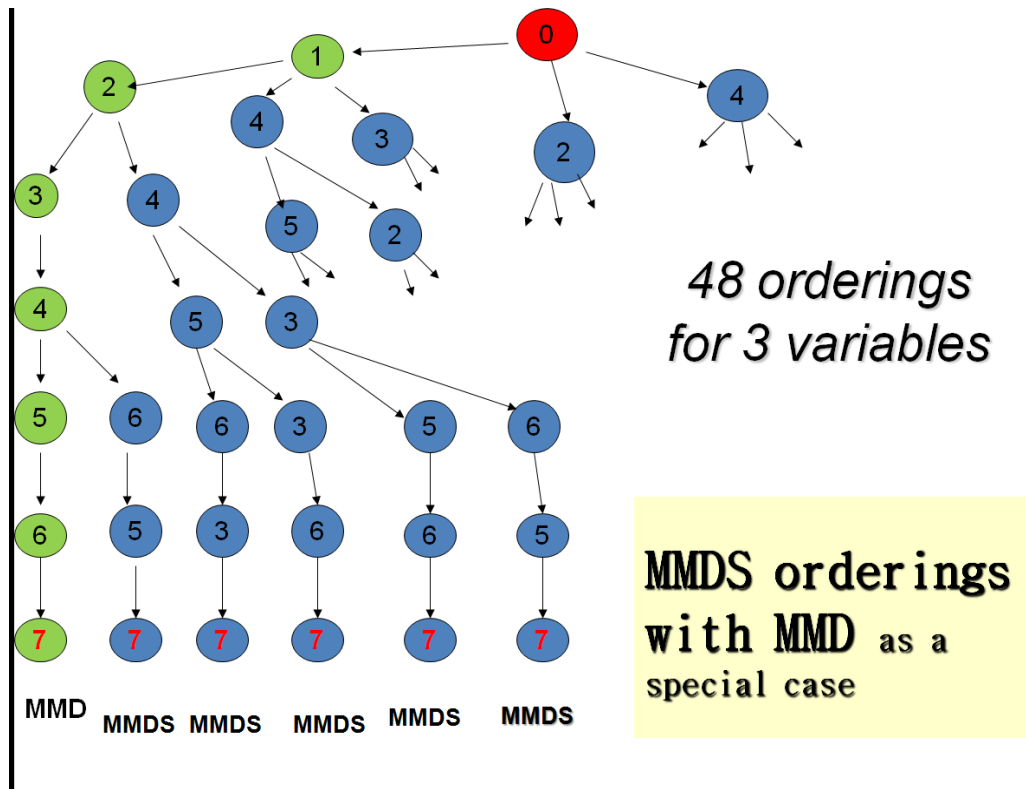
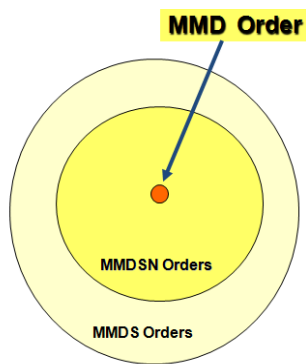


Figure 5.4.6. Graphical visualization of MMDS orderings in Karnaugh Maps.

**5 Concluding, MMD is just one order. MMDS is a very large set of orders that are correct and can be used in the synthesis. MMDSN is just some subset of MMDS selected to speed up synthesis and is easy to generate but not necessarily best in any sense (see Figure 5.4.7).**

**6 MMDSN does not include MMD. For instance for three variables MMDSN will be 000, 100, 010, 001, 110, etc, thus MMD is not generated as MMD has 011 immediately after 001 and 010. For better results we artificially add MMD to MMDSN in one variant of our software. This is illustrated in Figure 5.4.7.**

**7 Further research should be to find other sets of orderings included in MMDS but possible smaller and better than MMDSN.**



*Figure 5.4.7. Graphical visualization of MMDS orderings in Karnaugh Maps. MMD order was added to MMDSN to have results always not worse than MMD CAD tool.*

## 8 5.5. MP ALGORITHM

Earlier attempts to improve MMD algorithm resulted in less costly circuits for some functions and non-convergence for others [Kerntopf04, Khlopotine02, Mishchenko02, Stedman04]. Thus the order of selecting outputs to be transformed first was found experimentally to be more important than the heuristics to choose gates.

For larger number of variables, a variant of our algorithm was created based on the following principles:

- (1) Rather than maintaining a set of tables mapping inputs to outputs, the algorithm creates these columns implicitly, simulating minterms one-by-one. The simulator uses the equations from the specification, together with the part of the reversible circuit already constructed. To demonstrate the concept, imagine two circuits as in Figure 5.5.1 cascaded back to back and simulated from inputs at each stage of minterm transformation. The first circuit described by equations, represents the function under synthesis, and the second circuit is the outcome of synthesis (in reverse order of gates). When the synthesis process completes, two equivalent circuits are generated, one mirror of the other, where the first circuit is specified by equations, and the second by reversible gates in reverse order. When we simulate this

composed circuit, for every input minterm, the same minterm is obtained at the outputs of the concatenated circuits, and hence, the concatenated circuits are a reversible identity. Since the circuits mirror one another, the solution is represented by the second circuit of the two concatenated circuits. Figures 5.5.2 – 5.5.6 visualize stages of applying this algorithm.

- (2) A number  $k$  of randomly selected MMDSN orders is generated representing the function being synthesized. The solution with optimal cost is then selected with the possibility of backtracking if the temporary cost exceeds the minimum cost determined earlier in the process.
- (3) When possible, template matching method from MMD is used on the result for post-processing to further improve the quantum cost.

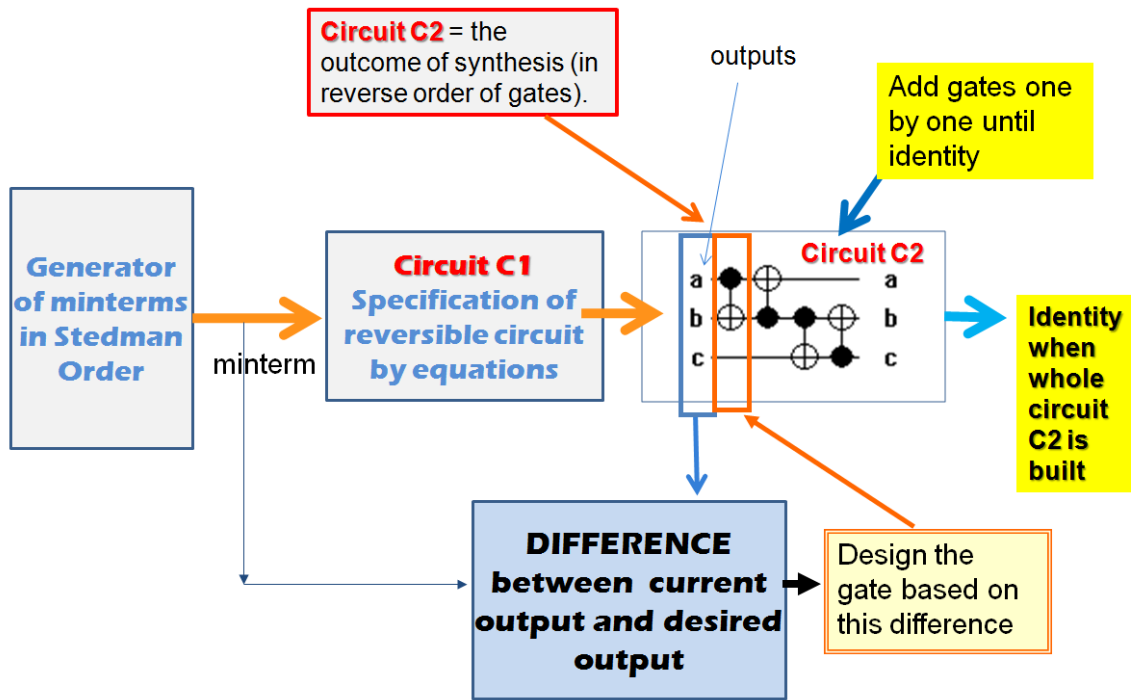


Figure 5.5.1. The Basic Scheme of the MP algorithm.

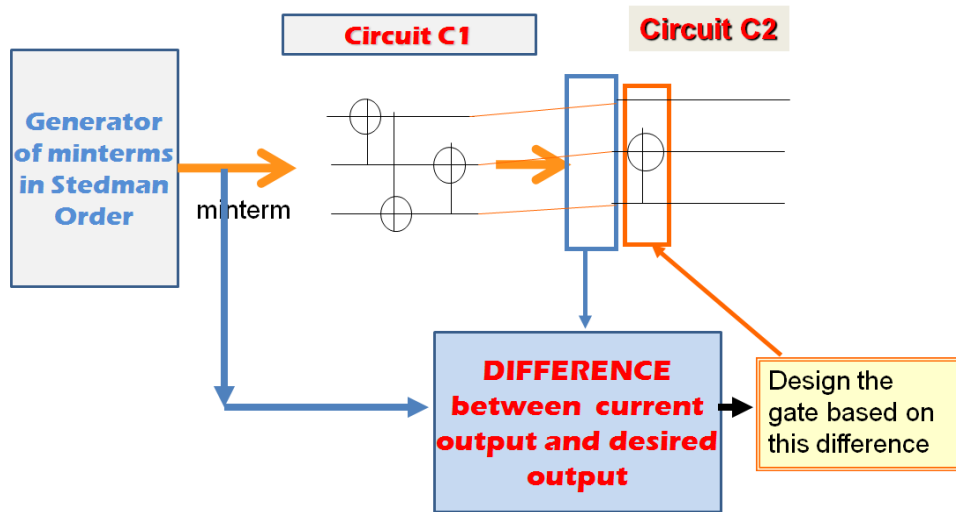


Figure 5.5.2. The visualization of the MP algorithm. First stage.

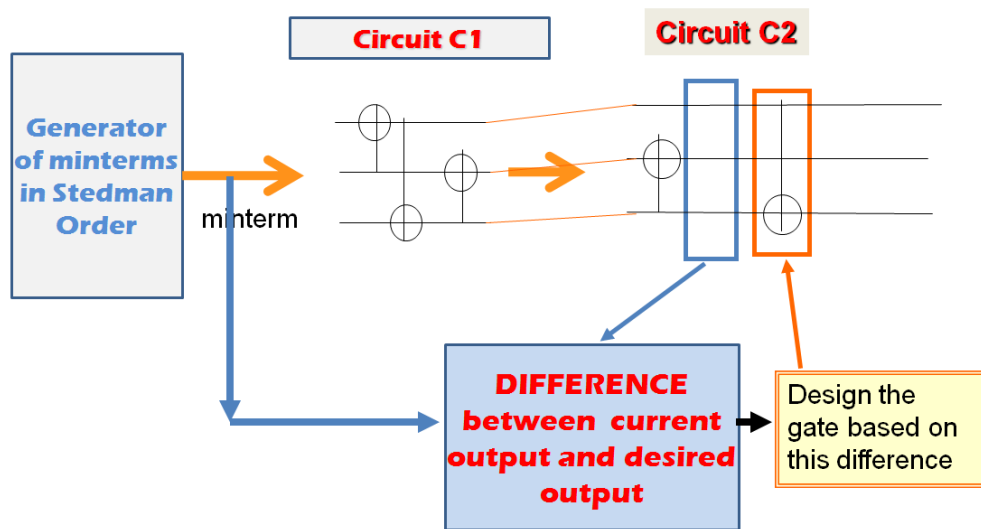


Figure 5.5.3. The visualization of the MP algorithm. Second stage.

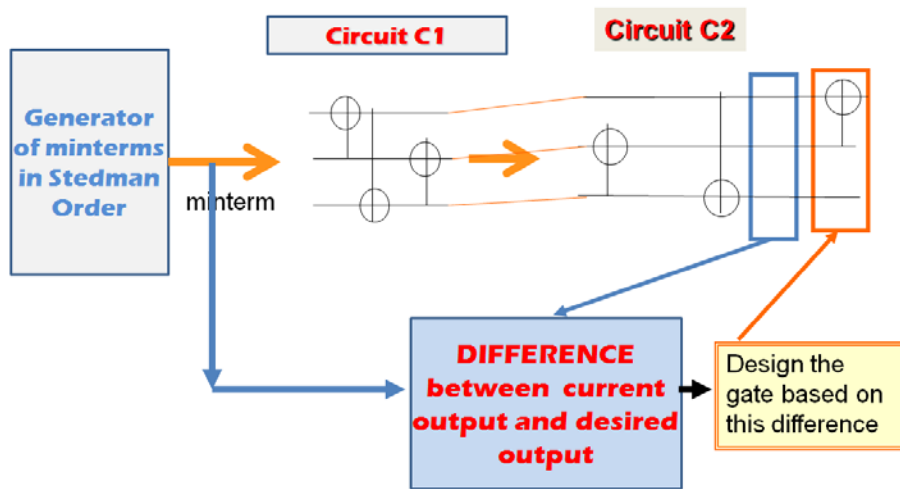


Figure 5.5.4. The visualization of the MP algorithm. Third stage.

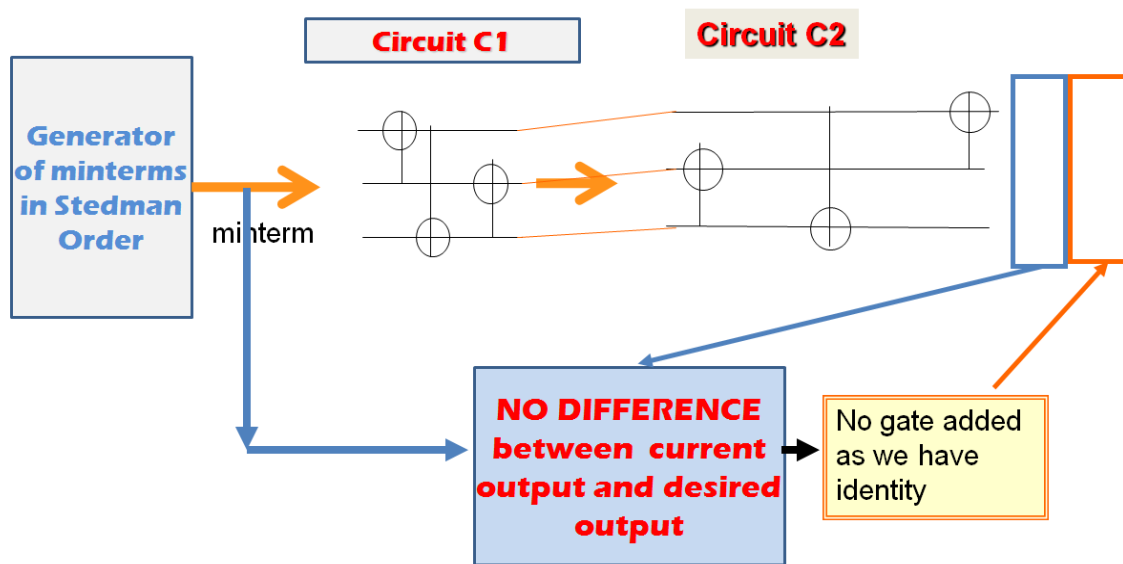
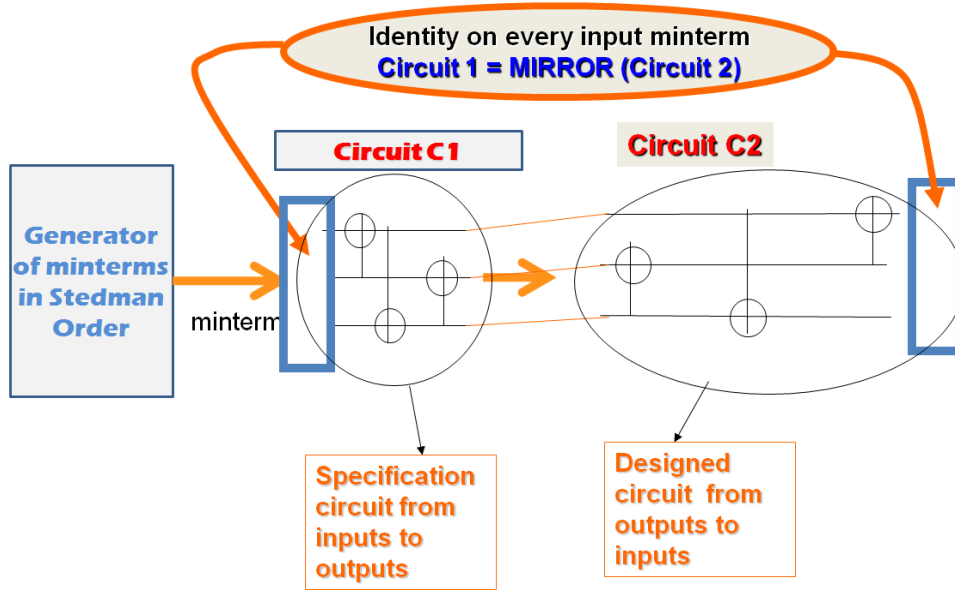


Figure 5.5.5. The visualization of the MP algorithm. Fourth stage.



9

Figure 5.5.6. The visualization of the MP algorithm. The final stage.

## 10 5.6. RESULTS OF MP FOR FOUR VARIABLES

For functions of four variables, we created a set of randomly generated four-bit reversible functions, AHP1 - AHP50, and synthesized them using the original MMD, MMDS and our MMDSN orders. For MMDS and MMDSN, we tested the AHP functions against all possible permutations and calculated the minimum possible quantum cost and gate count as shown in Table 5.1. It is evident that our selective orders consistently produce superior results compared to the single MMD order for a negligible time penalty. Notice, however, that although the MMDSN order did not generate the optimal gate count

generated by MMDS, the time advantage of MP is huge at 4 bits, and would be astronomical at greater number of bits. Even at higher number of bits, MMDSN order consistently produces better results than MMD within tolerable time. For example, at 11 bits, MP accrued a saving of 191 gates taking about six minutes to synthesize 5000 MMDSN sequences selected at random, and maintaining the solution with the best gate count and quantum cost.

Although the current implementation of the MP algorithm does not utilize parallel processing, the algorithm is well suited for parallelization through threading, multiprocessing or within a cloud infrastructure. Such capability would allow synthesizing circuits based on selecting a larger iteration variant ( $k$ ) and thus enabling synthesis of even larger functions. The reader should note that in this study, neither MMD or MP used local optimization techniques, e.g. template matching of MMD, which would ideally reduce the number of gates even further. Although MP would run even slower with template matching, its inclination to parallelization would easily minimize such an impact. An additional advantage of MP is that we can have a trade-off – the longer we run the new combined algorithm the better our result is. This trade-off property is missing in both MMD and Agrawal/Jha approaches.

## 11 5.7. RESULTS OF MP ALGORITHM FOR MORE THAN FOUR VARIABLES

Table 5.2 shows results for  $k = 5000$  produced with a single threaded application on a Windows 7 operating system running on Intel® Core™2 Duo 2.93 GHz processor. The application allows the user to set  $k$  to any value to get the trade-off between synthesis time and quantum cost improvement. We were not able to compare MP with original MMD on larger functions since MMD does not accept functions of 30 variables as it is not able to store a vector with  $2^{30}$  rows in memory. As we see in Table 5.2, the improvement here is best for functions with less than 7 variables, which means that value of  $k$  should be increased. If better results were published for new versions of MMD or other software, it will be our future research to compare them with MP.

To understand the limitation of our approach for very large functions, we created a sample reversible function, AHP30\_1, of 30 variables, which was input as a separate equation for each bit (this variant of MP is not format compatible with MMD and other programs). The synthesis generated a quantum array of 4496 gates and took 2 hours and 45 minutes to complete. The function was a simple cascade of Toffoli gates where each variable controls its immediate successor. Our choice of a simple representation of function, at this time, sets for us a foundation for future research as we plan on extending our method to other functions of 30 variables or more. Results cited in this chapter are currently available on <http://www.quantumlib.org:21012>.

## **5.8. DIFFERENCES BETWEEN MMD, MP AND OTHER ALGORITHMS.**

### **FUTURE RESEARCH ON IMPROVING THE CURRENT VERSION OF MP.**

1. After the circuit is built, the MMD algorithm needs the template simplification tool, to further reduce the cost of the circuit. In MP, we are performing the search for the best gate at every level, so we will not need Template Simplification tool. Template methods do not work well for functions of many bits and with many gates. Still they can be used in an attempt to improve on MP results.
2. MMD uses bi-directional approach, whereas our synthesis algorithms search always from output to input. Bi-directional search can be still added to MP on the same principles as to MMD.
3. Researchers from Markov's team ([Shende01, Shende02, Markov and Bullock] consider only 3 qubit circuits in synthesis and their optimal method takes several hours to optimize some 4 qubit circuits. This approach is thus completely impractical for more than 4 variables. The advantage of their approach is that only quantum realizable gates are assumed, and thus a 4 qubit Toffoli gate is not used as a building block in

synthesis. Their method is only for completely specified functions which are already made reversible. If the reversible function is odd, a single ancilla qubit is added to make it realizable in the cascade. Odd function has an odd number of minterms. The even function has an even number of minterms. MP with MDS orderings, produces minimal or nearly minimal results on several functions. It should be compared with Shende's approach, however the benchmarks from Shende are not available, as well as their source code, so the comparison would be difficult. A variant of MP can be created to be competitive with Shende's approach, but this is not high on research priority as having high quality but approximate results for large functions is more important than having exact minimum solutions for small functions.

4. MMD method can solve functions of 7 variables, but this method assumes Toffoli gates with many inputs, which are not directly realizable in quantum and may require many input constants or mirror circuits to realize. Initially the authors did not consider the high costs of using this assumption in quantum circuits. Their method is for completely specified functions only. Their solutions for several functions including the rd53 benchmark are not minimal. It is not clear when to add ancilla bits and how many are needed in their method. For example, on page 88 in [Maslov03b] there is an example of rd53 realization with 5-qubit Toffoli

gates, but additional constants for these gates are not shown. Each of these results can be matched or improved for any type of cost assuming we run MP long enough. Future research should compare all known benchmarks and experimentally find values of  $k$  that would be sufficient to outperform all other software on all examples in terms of circuit cost and size.

5. Recently, efficient group-theory based algorithms for synthesis of all 3-qubit functions have been presented (Guowu Yang, [Yang05, Yang06]), so the 3-qubit reversible circuit are no longer treated as a practical challenge in the reversible/quantum logic research community. Their basic algorithm synthesizes only 4-qubit, completely specified functions. In one more variant, MP can be compared with this software to find the value of  $k$  for MMDSN for which MP will outperform Yang's software. For all 3-variable functions, MMDS found nearly optimal results. The same experiment can be done for MP with MMDS orderings for 4 variable functions.

## **5.9. CONCLUSIONS ON MMD-LIKE ALGORITHMS VERSUS NEW SEARCH ALGORITHMS**

By MMD-like algorithms, I mean here algorithms that have no search and use the concept of minterm transformation.

### **5.9.1. How to improve on MMD for a small number of variables in functions.**

I presented in this chapter a new approach that was influenced by MMD and the work of PSU group that preceded MMD. Our new approach to synthesize binary reversible cascades for incompletely specified functions also results from this work, but is not presented in this chapter. It will be presented in chapter 6.

The main goal of Chapter 5 was to introduce various methods to improve the MMD algorithm. One area of improvement was focused on input order. It was found that the natural binary order is only special because it falls in the category of input orders that do not exhibit control line blocking. All orders of this type can be used with 100% convergence in MMDS. This added degree of freedom allows the MMDS program to run through an assortment of input orders until it finds the best circuit for that specific function. Again, different input orders are better suited for distinct functions, so cycling through all input orders will find the most optimal solution. As the number of inputs

increases, the number of input orders increase exponentially. In order to fully utilize this property of multiple input orders, an improved technique must be further developed to find a subset of all MMDS input orders other than MMDSN. For example, by sampling across all outputs then converging in the best area of input orders or by sampling output orders to focus the search on a specific area. Various ways of traversing Hasse diagrams can also be considered. This is an area of further research. Alberto Patino and Maher Hawash created already successful software that exceeds MP and is based on these new ordering ideas which were started in my research and presented in this dissertation.

Concluding, these are the most general new ideas:

1. Investigating various orders of variables; creating new orders that are different than MMD, MMDS or MMDSN.
2. Creating subsets of orders dynamically, adapting them to the given function.
3. Use some reduced cycle decomposition for some types of functions. It is well known that a reversible function can be represented by cycles, these cycles can be decomposed in various ways [Yang04]. The decomposition to cycles are much related to orderings. Decompositions allow to investigate some small local orderings dynamically selected rather some global non-adjustable orderings such as MMDSN.
4. Better heuristics should be used for searching trees of orderings.
5. More powerful gates can be used, such as multiple-controlled Fredkin, Kerntopf and other gates introduced in chapters 2 - 4.

I believe that by combining these ideas and implementing these ideas in a unified software, the MP software can be further improved.

These improvements will include also relational specifications, and don't cares in particular, which are presented in chapter 6. Some smarter search should be used. It may be a Genetic Algorithm or similar statistical approaches. Alberto Patino has already good results using the smart order generation approach. Other good candidate is the set of ideas that are based on group theory [Yang04] but these ideas must be rephrased to language and representation of Boolean and MV functions rather than groups. These are all areas of future research. Hopefully several PSU students already work on these ideas.

### **5.9.2. How to improve on MP on very large number of variables.**

In this section, I discuss new ideas for synthesis of reversible functions. These ideas were initially included into the dissertation but were next removed as they have been not experimentally verified. Here I briefly mention these ideas as an area of future research.

1. **Approaches based on adding ancilla bits.** One can introduce some limited number of ancilla bits. The minimum is one ancilla bit. The maximum is  $N$  ancilla bits for a function of  $N$  variables. Optimistically, due to the introduction of ancilla bits, the approaches based on adding dynamically ancilla bits are able to synthesize both incompletely and completely specified functions, reversible and

irreversible functions without adding any additional ancilla bits to make the function reversible. Given also the freedom of output permutation, we assert that the additional  $N$  ancilla bits introduced by this approach more than make up for themselves and present this as an alternative approach to ancilla bit limiting methods for binary reversible logic synthesis.

2. Some of these approaches are being programmed by Patino and Hawash with good results.
3. As most of these methods are not programmed yet, we do not know what is their real advantage but I believe that in worst case, we will be able to find some subareas of design where the new methods will be better than the MP algorithm, the Agrawal/Jha algorithm and all known approaches.
4. **More advanced benchmarking for large functions.** In the meantime, new authors entered the race to develop synthesis software for reversible circuits, but software from MMD and Agrawal/Jha is still best overall, so we should compare our new results with these two software packages. As of now, MP is the best algorithm in terms of size and cost, all new software should be benchmarked against MP as well.
5. **Synthesis methods based on Cycle Decomposition.** We still have to work on this method as it can be combined with our other methods. Ideally the software should try various strategies for each given function to minimize. I hope this will work especially when our software is tested on big functional specifications obtained from netlists, state machines or block descriptions of reversible logic

[Shigvand05] as a future work. We will have to develop and program completely new data structures for these variants which will be better than expressions.

6. **Synthesis methods based on Group Theory.** The Cycle Decomposition Method can be further enhanced by finding special cycles for which this method is suitable. For example, one may find that cyclic decomposition to cycles of certain length may give better synthesis results than to some other cycle lengths. This work requires more knowledge of group theory. The software and data structures should be flexible to provide support for such decompositions.
7. **Gates.** Incorporating better and new gates into the synthesis algorithm. In theory, any gate from chapters 2 and 3 can be incorporated, as the methods presented here are general.
8. **Local Optimizations.** Using MMD for optimization after using MP for synthesis. Adding better and bigger templates to the MMD template based reduction. This is currently being done by the original creators of MMD and their software can be merged with MP.
9. **ESOP Minimization.** Internally use two programs: MP and the ESOP minimizer Exorcism. Improve the ways to handle don't cares, possibly using DCARL software developed by Manjith Kumar or develop new software for this task.
10. **More cost functions.** Cost functions such as latency, testability, exact quantum cost (number of EM pulses, etc) can be added.

## 12 5.10. CONCLUSIONS ON THE WORK PRESENTED IN THIS CHAPTER

In this chapter we presented briefly the literature background on top algorithms for synthesis of reversible cascades with no ancilla bits. Our previous research and its drawbacks are also briefly mentioned. Finally, we presented the top achievement of this dissertation - a new algorithm MP to synthesize reversible circuits in the spirit of MMD. As the algorithm is a generalization of MMD, it can never create solutions worse than those by MMD. But it can create results of smaller cost and can find solutions to problems that are too large for MMD to synthesize. Our algorithm does not require to store the large truth table or other exponential representations as it calculates the values on the fly from the logic equations. MP scales better to large functions than any other available algorithm. It can solve 30 variables functions. Although MP still needs an exponential number of simulations, it does not need to store exponential data. Also we use many orders of minterm creation which leads to more efficient circuits. However, we pay the price of a slower synthesis process. The results are concluded in Table 5.3. The best algorithm for each property is made bold and underlined.

Table 5.3. shows that all these algorithms do not use ancilla bits, relational specifications and conversions from irreversible to reversible forms. The results of chapter 5 have been extended in chapter 6 to the synthesis of incompletely specified functions, circuits with

ancilla bits and irreversible specifications (typical for instance in quantum state machines [Kumar07, Kumar08]).

Property\algorithm	MMD	Agrawal and Jha	Shende et al	Yang et al	MP
Speed or time complexity	<u>Fast</u>	Fast or medium	Very slow	Very slow	Slow
Quality of results on quantum cost	Approximate on medium functions, no guarantee of exactness	Approximate on medium functions, no guarantee of exactness	Exact for small functions	Exact for small functions	<u>Top quality on medium and large functions, no guarantee of exactness</u>
Size of functions that can be processed	12	18	3	4	<u>30</u>
Scaleability of algorithm for large functions – space	weak	medium	very weak	very weak	<u>top</u>

complexity					
convergence	Yes, if truth table space allows, most examples are 8 bits.	Yes, if PPRM space allows, few examples more than 10 variables tested.	Only for functions of 3 variables	Only for functions of 3 variables, some special types of 4 variable functions	<u>Yes, if expression space allows, largest tested was 30</u>
Ancilla bits	no	no	no	no	no
Incomplete and relational specifications	no	no	no	no	no
Conversion from irreversible to reversible	no	no	no	no	no

**Table 5.3 Algorithm Comparison Table**

As the reversible logic is still a research topic rather than an industrial topic, the speed of synthesis is less important than exploring larger circuits and being able to decrease circuit

costs. The trade-off that exists in MP between the time and cost of solution helps in this research.

Function	MMDSN			MMD			MMDS		
	#	<i>Q-</i>	<i>Time</i>	#	<i>Q-</i>	<i>Time</i>	# Gates	<i>Q-</i>	<i>Time (ms)</i>
	<i>Gates</i>	<i>Cost</i>	<i>(ms)</i>	<i>Gates</i>	<i>Cost</i>	<i>(ms)</i>		<i>Cost</i>	
AHP 0	18	102	8.393	20	144	1.074	15	55	
AHP 10	16	68	6.991	29	209	0.022	14	42	
AHP 100	22	150	8.040	25	149	0.018	18	98	
AHP 102	21	109	7.653	28	192	0.019	19	103	
AHP 104	19	99	7.408	28	192	0.020	17	73	
AHP 106	21	129	7.567	24	116	0.016	17	77	
AHP 108	20	108	8.078	21	129	0.015	17	77	
AHP 100	16	80	7.497	19	111	0.014	14	54	
AHP 100	21	113	7.513	31	223	0.014	18	78	
AHP 100	20	136	7.056	23	167	0.029	15	79	
AHP 100	17	93	7.495	24	172	0.030	17	109	
AHP 100	19	95	6.682	31	215	0.024	18	90	
AHP 101	18	74	6.953	30	230	0.028	17	85	
AHP 101	23	131	7.146	28	168	0.031	18	70	
AHP 101	23	139	8.069	27	179	0.031	19	75	
AHP 101	18	126	6.748	23	167	0.030	15	79	
AHP 101	17	105	6.939	25	197	0.030	15	63	
AHP 102	18	106	7.317	25	193	1.803	16	96	
AHP 102	19	111	7.697	24	156	0.153	14	54	
AHP 102	22	138	6.622	30	218	0.148	16	76	
AHP 102	14	66	7.252	17	113	0.154	14	66	
AHP 102	14	86	7.343	20	148	0.157	13	81	
AHP 103	21	137	7.776	27	167	0.124	16	80	
AHP 103	20	108	6.726	27	187	0.106	17	93	
AHP 103	19	123	7.132	22	138	0.102	15	71	
AHP 103	19	107	7.257	26	186	0.093	17	81	
AHP 103	18	106	7.927	18	106	0.083	13	65	
AHP 104	16	96	6.478	22	174	0.078	11	39	
AHP 104	22	146	7.263	25	173	0.080	19	99	
AHP 104	19	107	7.325	23	159	0.096	16	92	
AHP 104	19	107	7.739	23	147	0.092	15	71	
AHP 104	18	94	6.484	20	120	0.096	17	89	
AHP 105	23	123	7.325	34	230	0.083	19	83	
AHP 105	18	110	7.557	26	166	0.080	16	84	
AHP 105	17	81	7.226	24	164	0.047	16	76	
AHP 105	17	93	7.757	28	196	0.813	15	67	
AHP 105	18	118	6.991	23	155	0.015	15	55	
AHP 106	19	151	8.110	21	161	0.019	15	83	
AHP 106	19	107	7.268	31	247	0.020	16	76	
AHP 106	23	131	7.357	29	189	0.017	20	84	
AHP 106	18	122	7.055	31	235	0.017	15	75	
AHP 106	22	134	8.606	21	97	0.017	19	99	
AHP 107	18	106	7.707	22	158	0.018	16	80	
AHP 107	20	112	7.611	23	159	0.019	16	72	
AHP 107	22	126	8.236	26	194	0.017	18	106	
AHP 107	21	121	8.644	28	184	0.020	18	74	
AHP 107	21	105	7.690	30	222	0.021	18	78	
AHP 108	21	145	7.879	21	109	0.016	17	93	
AHP 108	21	133	8.109	29	233	0.016	16	92	
AHP 108	23	119	8.797	31	187	0.014	20	104	
AHP 108	20	116	7.367	27	195	0.014	17	85	

Table 5.1. Comparison of MMD, MMDS and MMDSN orders on 50 random functions of 4 variables.

# bits	Function	MMDs # of gates	MMDs qcost	My # of Gates	My Qcost
4	ou4_4file	30	162	25	85
4	ouhwb4file	24	154	18	58
5	ouhwb5file	64	914	49	421
6	ouhwb6file	162	4036	139	2341
7	ouhwb7file	374	13893	327	10007
7	ouham7file	324	13145	302	10683
8	ouhwb8file	995	58605	960	45273
9	ouhwb9file	2249	188997	2140	144481
10	ouhwb10file	4953	538588	4836	463578
11	ouhwb11file	10929	1412439	10908	1309776

Table 5.2. Comparison with MMD's number of gates and quantum cost.

## **Chapter 6: SYNTHESIS OF REVERSIBLE CASCADES FROM INCOMPLETE AND RELATIONAL SPECIFICATIONS**

This chapter presents several methods to synthesize reversible cascades from various specifications. These specifications are:

1. Functions with don't cares
2. Relational specification that are more general than functions with don't cares
3. Irreversible specifications, which are converted to relational specifications and next to reversible circuits.

These kinds of specifications appear , for instance, in quantum state machine design, so they are practically important in circuit optimization.

### **6.1. A SYNTHESIS METHOD TO SYNTHESIZE REVERSIBLE FUNCTIONS WITH NO ANCILLA BITS USING ARBITRARY GATES AND RELATIONAL SPECIFICATION.**

#### **Multi-input multi-output binary relational synthesis**

In this section, I present a new synthesis method for incomplete specifications and arbitrary permutative gates. We call this method, the “*Symbolic Matrix Method*”. This is

a tree search method, which means, the circuit is created from outputs to inputs, as in previous chapters. At every stage the algorithm creates several nodes in a tree called *successor nodes* of the node from the previous level of the tree. Selection of each of these nodes represent the choice of some gate from the library. The initial node represents the function to be synthesized. The intermediate nodes correspond to some partial subfunctions of the function to be synthesized (and also to corresponding partial circuits realized so far). Each branch of the tree that terminates with node being an identity function is a solution. In this sense, the methods from this chapter are similar to the methods from previous chapters. The basic similarity is the step-by-step construction from outputs to inputs. We can create several variants of this basic heuristic - we can find equivalent methods to synthesize “from relational specifications” that are counterparts of most synthesis methods presented so far. This relates to both, tree-search methods such as Greedy and local selection algorithms such as MMD or MP.

The method presented in this section uses arbitrary permutative matrices to represent reversible gates. Each gate is a permutative matrix, and composition of gates corresponds to the product of these matrices in reverse order. The question is how to select gates so that the composition of their matrices is equal to the specification matrix.

The synthesis problem for relational specification is formulated as follows:

1. Given is an incomplete specification in the form of an incomplete permutative matrix  $U$ , meaning a matrix with symbols “0”, “1” and “-“. This matrix could

become a standard permutative matrix (single “1” in every row and in every column, all other symbols are “0”s) if one would substitute all the “-“ symbols accordingly. We assume that such substitution always exists, and that this substitution is (usually) not unique. This is the assumption of this method – this assumption allows to create a relatively efficient algorithm. The cases that such substitution does not exist will require different methods, and these methods will be discussed in next sections.

2. Given is library LIB of gates (permutative unitary matrices) that will be used in the synthesis. These gates are described by matrices of size  $2^k * 2^k$  where  $k$  is the width of the cascade. We say therefore that the gates are “for the whole width of the cascade”. These gates are the same type of gates as discussed in Chapters 4 and 5. This approach simplifies the synthesis, but is not applicable to large cascades, because the size of the library would become prohibitively large.
3. Find the cascade C of gates from the library LIB that realizes the given incomplete specification. This means that cascade C corresponds to a permutative matrix  $M(C)$  such that matrix  $M(C)$  matches matrix U on all cares, i.e. for every symbol “0” or “1” the matrices U and  $M(C)$  are the same. At the same time, the symbols from  $M(C)$  which are not 0 or 1 are replaced with binary symbols which correspond to the circuit being realized.

### Example 6.1.1.

Given is the relational specification of the cascade in the form of an incomplete permutative matrix U.

$$U = \begin{array}{c} \begin{array}{cccc} & 00 & 01 & 10 & 11 \\ \begin{array}{c} - \\ 0 \\ - \\ 0 \end{array} & \begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \end{array} & \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \end{array} & \begin{array}{c} - \\ 0 \\ - \\ 0 \end{array} \\ \begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array} \end{array} = \begin{bmatrix} x & 0 & 0 & y \\ 0 & 0 & 1 & 0 \\ u & 0 & 0 & v \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

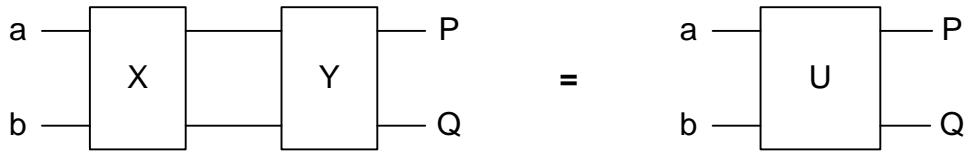
*Figure 6.1.1. Incomplete Permutation Matrix for example 6.1.1.*

The first step of the algorithm is to replace every symbol of don't care , i.e. “-“ , with a unique symbolic variable. This is shown at the right side of Figure 6.1.1. Four variables are introduced, one for each symbol “-“. These are variables x, y, u and v.

The next stage of my algorithm is to assume that our relational specification U (matrix U which formally is not permutative at this stage) is a composition of a library gate (matrix) Y and the remainder circuit (matrix) X. Because we synthesize the cascade from outputs to inputs, gate Y is at the right and the remainder circuit is X at the left. Gate Y is a specific gate from the library, so it has a definite permutative matrix Y. There exist a finite library of gates, each specified by its standard (completely specified) permutative matrix, this is similar to the method from chapter 5. Observe however that the matrix of

the remainder circuit  $X$  is a relational specification, so it still has unknown (symbolic variables) in its matrix and is not a standard permutative (binary) matrix. The goal is now to convert this matrix to a permutative matrix in our process of reversible circuit synthesis.

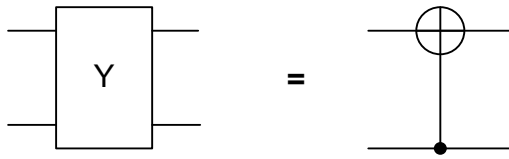
Based on the above explanation, we assume the general decomposition setup as one shown in Figure 6.1.2. Here  $Y$  is the selected gate from LIB (the gate is the same as its permutative matrix).  $X$  is the remainder (residue) reversible function (circuit, symbolic specification, etc). Matrix  $X$  results from selecting gate  $Y$  for the initial function specification  $U$ . From now on, we will use the name or the matrix representing this gate interchangeably.



*Figure 6.1.2. General Decomposition of Unitary Matrix  $U$  into a sequential composition of blocks  $X$  and  $Y$ . Relational specification  $U$  is decomposed to a composition of gate  $Y$  that comes from our library of gates  $LIB$  and the remainder specification  $X$ . This procedure is repeated by decomposing  $X$  in the same way iteratively, until the remainder  $X_n$  is found that has a matrix that can be completed to an identity matrix. This completes the synthesis.*

Remember that gate Y is only one of the gates in the library. The algorithm can be designed as a tree-search algorithm that at a given depth of the tree, can make one of three choices: (1) selects an arbitrary gate, (2) selects all gates from the library (branching for the depth-first or breadth-first search variant), or (3) selects some subset of available gates that optimize some cost function, such as a Hamming Distance. Any method presented so far can be adapted to these types of tree-searching algorithm. Therefore, in general, the method introduced in this section is the “ tree search method”, and we show only few branches of this tree in our explanations here.

Assume that in this example the algorithm selected gate Y which is shown in Figure 6.1.3 below:



*Figure 6.1.3. Realization of gate Y from library of gates LIB. A Feynman Gate with Exor Up was assumed here as gate Y. The library stores every gate in the form of its permutative matrix, name and schematics.*

This means, the algorithm selected CNOT gate with EXOR up as the gate from library LIB. Please note that LIB is a library of arbitrary cells. It is not a set of only Toffoli and Feynman gates (with EXORs in various qubits), as was presented in chapter 5. The library must be though a library that is universal, which means, that any function can be realized with gates from this library. In one variant of the algorithm, not discussed in this thesis, we assume that the ancilla bits can be used. In the algorithm presented in this section, however, the ancilla bits are not used.

Thus the permutative matrix  $Y = Y^{-1}$  of the gate from Figure 6.1.3 is shown in Figure 6.1.4:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

*Figure 6.1.4. Permutative Matrix  $Y = Y^{-1}$  Please note that the matrix from the library LIB and its inverse matrix are the same. This is the property of the gates that we use.*

At this point, having the specification matrix U and the matrix of the selected gate Y, it is possible to use matrix calculus to find the matrix of the reminder circuit X. This matrix is a symbolic matrix, not a permutative matrix yet.

Now I will prove mathematics of my method:

Assume  $Y * X = U$

Therefore, multiplying both sides of the equation by  $Y^{-1}$  we obtain

$$Y^{-1} Y X = Y^{-1} U$$

Therefore we obtain  $X = Y^{-1} U$ . Thus we obtain matrices as in Figure 6.1.5.

$$X = Y^{-1} U$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} x & 0 & 0 & y \\ 0 & 0 & 1 & 0 \\ u & 0 & 0 & v \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} x & 0 & 0 & y \\ 0 & 1 & 0 & 0 \\ u & 0 & 0 & v \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

*Figure 6.1.5. Synthesis based on matrix multiplication starting from incomplete specification  $U$  for example 6.1.1. The algorithm found the relational specification of the (new) remainder function  $X$ .*

Therefore, by applying gate  $Y$  to specification  $U$  and starting from outputs, the algorithm creates the relational specification of the remainder circuit – the algorithm obtains it in the form of matrix  $X$ .

Now the algorithm has either to complete the search in this branch of the search tree, or to find a new gate, Y1, for the decomposition explained above. Next the algorithm continues this process iteratively.

To complete the specification matrix X to a permutative matrix, the algorithm uses the specific heuristic. This heuristic is -- “increase the number of 1’s on the matrix diagonally”. This heuristic means increasing the number of self-mapping minterms which also (approximately) reduces the total Hamming Distance. This means that the variable symbols (x, y, u and v are our variable symbols in this example) should be substituted in the symbolic matrix with care (binary) symbols “0” and “1” in such a way that the number of symbols “1” on the diagonal of the matrix should be as high as possible. This heuristic method applied to the above matrix X produces the chain of substitutions of binary values to symbolic variables as below:

$$x = 1 \rightarrow y = 0 \rightarrow u = 0 \rightarrow v = 1$$

In general, to solve the problem of assigning binary values to symbolic variables, a heuristic tree search is executed with subsequent substitutions of binary values 0 and 1 to variables x, y, u and v, in a method that is similar to solving SAT (satisfiability) in branching algorithms.

Coming back to our example, using the heuristic of “ones on the diagonal” and no search, the algorithm creates a permutative matrix  $X$ . Now the search is done in the library LIB whether there exists a gate which is described by this matrix  $X$  or not. If such a gate exists, it is applied in the circuit. If an identity matrix is created, the algorithm terminates. If a matrix other than identity matrix is created, the next decomposition occurs and the algorithm iterates.

In the case of our example, as a result of a chain of substitutions, the matrix  $X$  from Figure 6.1.6a is found. The library LIB is searched and  $X$  is found as gate  $X' = \text{“CNOT with EXOR down”}$ . Thus matrix  $X$  and its corresponding circuit are shown in Figure 6.1.6a,b. The final solution, a reversible cascade corresponding to  $U$  is shown in Figure 6.1.6c. Observe that at this stage of synthesis, matrix  $X$  was recognized as a library gate  $X'$ . Otherwise, if the matrix would not correspond to any gate in the library, other binary substitutions would be tried. If search was not successful (no library gate would match), the matrix  $X$  would be further decomposed, as explained above. The decomposition will continue until the identity node in the search tree is found and thus the cascade is completed.

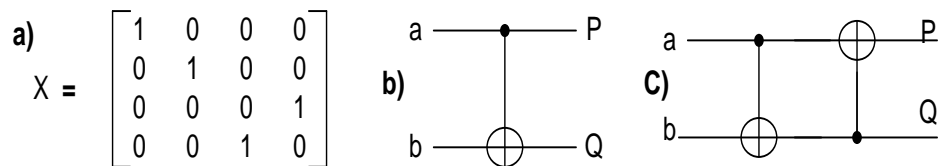


Figure 6.1.6. (a) The Unitary matrix of gate  $X'$  found by the algorithm, (b) the circuit corresponding to the unitary matrix  $X$ , (c) the complete circuit synthesized for the original specification  $U = YX$ . The circuit from Figure 6.1.6c corresponds to the general decomposition scheme from Figure 6.1.2.

Observe also that the solution to symbolic matrix  $X$  like one from Figure 6.1.5 is usually not unique, and we can find many solutions based on the tree search and the search heuristics proposed above. Another permutation matrix solution to symbolic matrix  $X$  from Figure 6.1.5 is the following:

$$y = 1 \rightarrow x = 0 \rightarrow v = 0 \rightarrow u = 1$$

which transforms symbolic matrix  $X$  to the following permutative matrix  $X''$  from Figure 6.1.7:

$$\begin{array}{cccc|cc} & 00 & 01 & 10 & 11 & & \\ \left[ \begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right] & & & & & \begin{array}{l} 00 \\ 01 \\ 10 \\ 11 \end{array} \end{array}$$

Figure 6.1.7. Permutative matrix  $X''$  for the symbolic (relational) matrix  $X$

The above matrix  $X''$  corresponds to the library gate  $X''$  (circuit from Figure 6.1.9). This circuit is explained in Figure 6.1.8. KMap for PQ is created directly from matrix  $X''$  in Figure 6.1.7 and next separated to KMaps for P and Q.

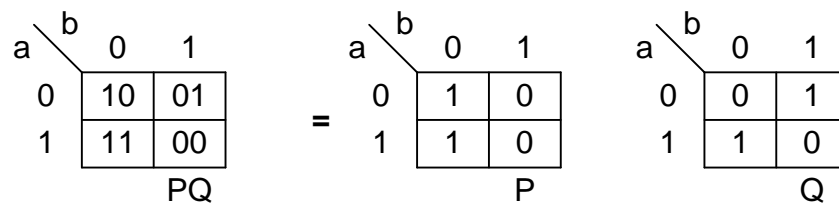


Figure 6.1.8. KMap for PQ, first together, then separated to KMaps for P and for Q.

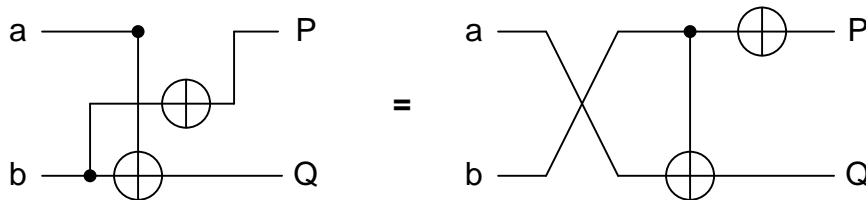


Figure 6.1.9. Second solution circuit obtained from the symbolic matrix in Figure 6.1.5.

In this example the algorithm found thus two solutions, the cascades from Figures 6.1.6c and 6.1.9. Each of these circuits corresponds to a branch of the search tree to one of its terminal nodes being an identity. There are several other branches in the complete search tree for our library which were not shown to reduce the explanation.

In general we have two tree search processes in our algorithm:

1. One tree search process solves the equations with symbolic variables, and creates a permutation matrix from a symbolic matrix,
2. The second tree search process selects gates Y for decomposition as illustrated above. The methods from previous chapters can also be used at this step.

As we have seen in Example 6.1.1, by representing the data in the form of symbolic matrices we generalize and reuse the methods that were developed and explained in previous chapters.

Few points should be emphasized:

1. This is a **search algorithm**. Therefore all issues related to search algorithms such as depth-first versus breadth-first - versus greedy search - versus heuristic cost functions and heuristic quality functions -- should be discussed. These discussions are omitted here as the algorithm was not implemented.
2. The **convergence of the algorithm** (i.e. the ability to find any solution for the specification) depends on the search method used and the library of gates that is applied. The realization of a similar algorithm developed by me was not always convergent for certain library that I used. However, the algorithm is convergent under assumptions of certain circuit length (number of gates) and certain library choice. This algorithm can be made practically convergent for all  $4 \times 4$

specifications. The convergence on larger functions depends significantly on the percent of don't cares in the initial specification and the placement of these don't cares in the matrix, so only the experimental analysis will help to evaluate the practical convergence on a set of benchmark functions.

It can be observed that the method from this section will work even if the **unitary matrices are not permutative but arbitrary unitary matrices** such as those of gates square-root-of-NOT. But this approach to synthesis is not discussed in this chapter, as this variant of method with square-root-of-NOT controlled gates has a large library of cells and thus becomes not efficient.

## **6.2. NEW HEURISTIC TREE SEARCH METHOD FOR INCOMPLETE IRREVERSIBLE SPECIFICATIONS: “THE MULTI-PARAMETER SEARCH”.**

The MPS algorithm presented in section 6.2 has two stages and several variants as it can process reversible and irreversible, completely specified and incompletely specified functions. The first stage of the MPS algorithm is to convert any type of specification, reversible or not, to an incompletely specified function. This stage is called the MI-MO-

IR algorithm. The next stage synthesizes the incomplete specification or converts it to a complete specification.

The MI-MO-IR algorithm preprocesses an irreversible incomplete function to a form that is next used for search in MPS. Its goal is to create incomplete functions of some special type.

Let us observe first that the irreversible functions, whether completely or incompletely specified, are of two types:

1. Those that can be completed to a completely specified function without adding at least one ancilla bit. We call these functions “backtrack-completeable”. Such functions are processed by MPS.
2. Those that cannot be completed to a completely specified function without adding at least one ancilla bit. We call these functions “ancilla-completeable”. Such functions are used in MI-MO-IR to convert them to the “backtrack-completeable”. form applicable by MPS.

Our entire method has the following stages illustrated in a simplified way in Figure 6.2.1.:

1. Complete each individual output to a simple balanced function, if possible.

2. Check if the set of individual outputs is reversible or can be completed to a reversible function without adding ancilla bits.
3. If necessary, select other completions of individual outputs to balanced functions and repeat step 2.
4. If it is not possible to find a set of individual outputs that can be completed to a reversible function, add a single ancilla bit.
5. If function is still not reversible, add one more ancilla bit.
6. Repeat step 5 until function is reversible.
7. When function with added ancilla is reversible, realize it using the second stage algorithm MPS. The second stage algorithm can use for each individual output either the original function or a completed function. The algorithm used at this stage gives the warranty that at least one completely specified function is found and realized. The selections of output variants can only decrease or increase the final solution cost. The costs of several solutions created by various variants of the algorithm are compared.

Remarks and details.

- 1) Our entire MPS method operates on (irreversible or reversible) functions with don't cares, this is similar to the method from section 6.1 but while functions were multi-output in section 6.1 and had the same number  $n$  inputs and outputs, in this section the specification allows for more freedom to describe individual don't

cares or relations for functions or their groups. The MPS algorithm processes therefore the most difficult and general types of functions to be realized in reversible circuit – incomplete irreversible functions that require adding ancilla bits to make them synthesizable.

- 2) The specifications for the second stage can be treated as reversible functions in which some binary symbols 0 or 1 were replaced with don't cares or relations. This way, the second stage of the algorithm has a warranty that it will find some solution, although it may require backtracking.
- 3) Our method of completing in step two is a search method. Therefore, in theory, any existing search algorithm can be used.
- 4) Our method uses ESOP minimization algorithm (Exorcism of Mishchenko) to represent and reduce the internal intermediate data,
- 5) The first stage of the method realizes outputs sequentially, which means group after group. In extreme case group-by-group means one output at a time. The method minimizes the number of ancilla bits, to zero in the case of specification is of type 1, and can be as high as  $m$  in case of arbitrary irreversible specification (this results from the well-known theorem that an arbitrary function with  $n$  inputs and  $m$  outputs can be realized in a reversible circuit with  $n+m$  bits of which  $m$  is the number of ancilla bits and  $n$  is the number of garbages).
- 6) The discussed algorithm allows to change the order of output bits. This property reduces (in some cases) the cost of the circuit. Observe that in some problems the order of outputs is arbitrary, but in – other problems, like quantum oracles, the

order is not arbitrary. Therefore there should be two kinds of synthesis methods for these two kinds of specification types. This idea is new and it has been not discussed in literature.

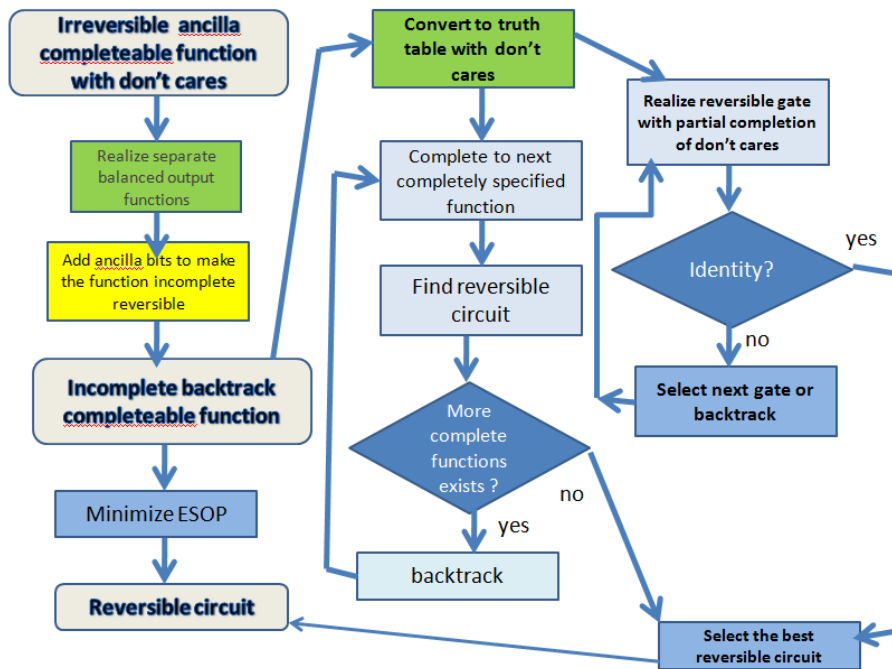


Figure 6.2.1. Main Idea of MPS algorithm to synthesize incompletely specified functions as a reversible cascade with (potentially) ancilla bits. The first step converts function to backtrack completeable function. Such function is realized with ESOP minimizer for small functions. For larger functions the design is converted to completely specified function or is realized gate by gate with partial conversions of don't cares to cares.

## 6.2.1. MI-MO-IR -- FAST SYNTHESIS METHOD FOR IRREVERSIBLE FUNCTIONS WITH DON'T CARES

### Example 6.2.1.

Synthesizing function from Figure 6.2.1.1a with as few ancilla bits as possible.

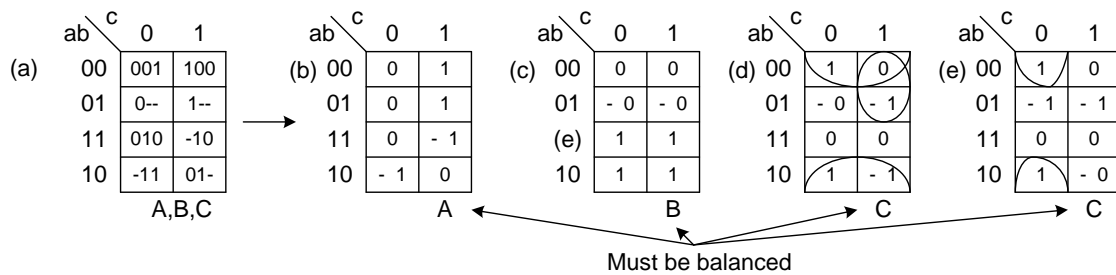


Figure 6.2.1.1. Synthesis of an incompletely specified 3\*3 function as a reversible cascade using preprocessing algorithm. Separating the outputs of the KMap to individual outputs A(a,b,c), B(a,b,c) and C(a,b,c), and using the property of balancedness for each of the outputs to replace don't cares with cares. (a) The initial specification of the problem with don't cares, (b) Separating the outputs of the KMap to individual outputs A(a,b,c), don't cares on left for minterms 100 and 111 are finally replaced with 1's shown at right in cells 100 and 111, (c)



1. The original multi-input multi-output incomplete function specification (Figure 6.2.1.1a, example with inputs  $a$ ,  $b$ ,  $c$ ) is separated to individual KMaps of individual output functions  $A(a,b,c)$ ,  $B(a,b,c)$ , and  $C(a,b,c)$ . In every function  $A(a,b,c)$ ,  $B(a,b,c)$ ,  $C(a,b,c)$ , the don't cares are converted to cares to make all these functions balanced (Figure 6.2.1.1 b, c, d, e). Observe that this is done for each function separately, possibly in various output orders.
2. ESOP minimizer program or any other algorithmic minimization methods is used to simplify these individual functions. In one variant, a function is converted to a balanced function first, and next it is minimized. In a new research variant these two stages can be combined.
3. The functions are synthesized one-by-one in a selected order, from the simplest functions to the most complicated functions. Functions that depend on less arguments and have less product terms and literals are considered to be the simplest. Any cost function can be applied.
4. Groups of incomplete functions can be analyzed and completed together to become completely specified functions. This is done in order to reduce the number of ancilla bits (Figure 6.2.1.2). These new functions are next minimized one-by-one, as discussed above.
5. Similar to all methods from this dissertation (except of MMD, MMDS and MP), this method searches a tree space of partial mappings and selected gates (like the the backtracking tree-search methods). The backtrack occurs however when no

sub-function (function) that is balanced (or reversible) can be found. This way, the algorithm prevents creation of too many ancilla bits.

ab \ c		0	1	
		0	1	
00		1	0	C
01		-	1	
11		0	0	
10		1	-	

Figure 6.2.1.3. Example for synthesis with don't cares using the MI-MO-IR algorithm. Illustrates realization of function  $C(a,b,c)$ . From separation pairs in Figure 6.2.1.2, the cell 010 should be 0 and the cell 101 should be 1.

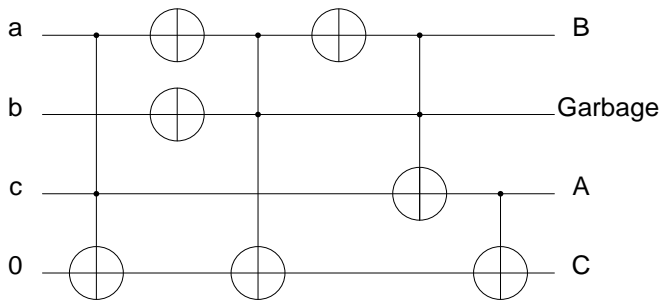


Figure 6.2.1.4. Reversible Cascade with one ancilla bit and one garbage bit synthesized for specification with don't cares and using the MI-MO-IR algorithm in which ESOP-based synthesis is internally executed. Output function  $B = a$  was realized as the first output bit. Output function  $A = a b' \oplus c$  was realized as the second. Creation of this function required creating of

garbage bit  $Garbage = b$ . Function  $C = a'b'c' \oplus ab' \oplus a'bc$  was minimized as the third output function. Observe that the created earlier output  $A$  was reused to minimize function  $C$ . This is one of main advantages of this method.

ab \ c		0	1
00		1	0
01		0	1
11		0	0
10		1	1

$C$

Fig. 6.2.1.5. Synthesis with don't cares using the MI-MO-IR algorithm in which ESOP-based minimization is internally used.  $C = A \oplus (a'b' \oplus ac)$

The function from Figure 6.2.1.1 is realized as follows:

1. The specification with don't cares is separated to individual output functions  $A(a,b,c)$ ,  $B(a,b,c)$ ,  $C(a,b,c)$
2. Based on simplicity and balancedness of functions  $A$ ,  $B$ ,  $C$ , the order of realizing these functions is decided by the algorithm. In this case, the order is  $B$ ,  $A$ , and  $C$ .
3. Using the ESOP minimizer, functions  $A$ ,  $B$ , and  $C$  are realized in this order ( $A,B,C$ ). Realization of each function uses not only primary inputs, but also all the realized subfunctions, such as the already realized output functions. Thus output  $A$  is used as an argument input variable while realizing  $C$ .

4. Subsets of output functions may be realized jointly to satisfy the reversibility condition to minimize the number of ancilla bits, Fig. 6.3.3. The complete function  $ABC(a,b,c)$  must be reversible so we use Figure 6.3.3 which shows the cells that have the same binary numbers in Figure 6.3.3 and need to be separated using function  $C(a,b,c)$ . This adds care minterms to  $C$ .
5. After completing the values of don't cares in output  $C$  according to Figure 6.3.4, function from Figure 6.3.5 is obtained. The solution  $C = a'b'c' \oplus ab' \oplus a'bc = A \oplus (a'b' \oplus ac)$ .
6. The final solution is given in Figure 6.2.1.4. As we see, only one garbage bit was added. The order of outputs was changed. (If necessary, the order of inputs can be changed again to the original order by using Swap gates).
7. Another circuit synthesized for the function from Example 2.1 is presented in Figure 2.6c. This circuit was realized as follows:
  - Function  $B = a$  was calculated as before.
  - Function  $A = ab = c$  was synthesized based on single-output balanced function.
  - Truth table of function  $C$  is calculated as  $c \oplus b'$ .
  - Function  $C$  is realized with reversible gates. This requires reuse of variable  $c$  which is not available so it is copied creating a garbage bit  $G$ .
8. Truth table of new reversible function is compiled from the single-output functions as in Figure 2.7. This table is a starting point to a new synthesis process using MPS. See section 3. Table in Figure 2.7 is backtrack-completeable, all

minterms not listed as rows in the table have all output don't cares for B, C, A and G.

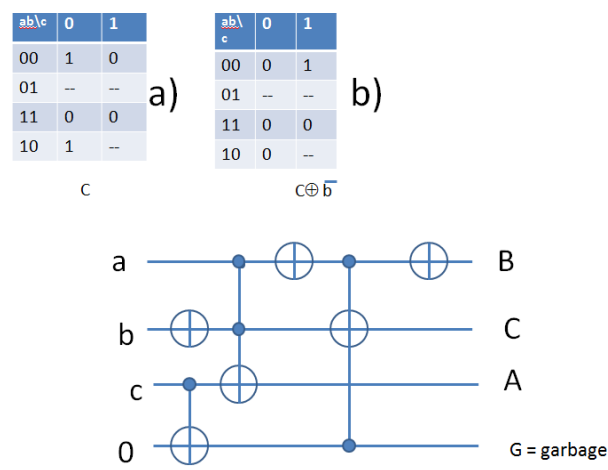


Fig. 6.2.1.6. Synthesis with don't cares using the MI-MO-IR algorithm in which ESOP-based minimization is internally used. In this variant, in contrast to the variant discussed above where there was no order of outputs , we assume order of outputs B,C,A,G (G is a garbage). This leads to the necessity of calculating new truth table of function A, as shown in Figure a) and b). The function B cannot be completed without copying variable c, so one ancilla bit initialized to zero is added and used to create input for Toffoli gate in C. This way, the garbage  $G = c$ , which is a common situation that inputs are copied to make function reversible.

a	b	c	c'	B	C	A	G
0	0	0	0	0	1	0	-
0	0	1	1	0	0	1	-

0	1	0	0	-	-	0	-
0	1	1	1	-	-	1	-
1	0	0	0	1	1	-	-
1	0	1	1	1	-	0	-
1	1	0	0	1	0	0	-
1	1	1	1	1	0	-	-

*Figure 6.2.1.7. Truth table for function from example 6.2.1 with assumed ordering of output variables BCAG, where G is a garbage bit.*

## **6.2.2. NEW HEURISTIC TREE SEARCH METHOD FOR INCOMPLETE SPECIFICATIONS BASED ON THREE HEURISTICS: “THE MULTI-PARAMETER SEARCH”.**

At this point, function has been converted from irreversible (complete or not complete) to an incompletely specified backtrack-completeable function. This means that the algorithm has the guarantee that it can complete a function to a correct reversible function without adding ancilla bits but possibly with backtracking.

A new method “multi-parameter search” (MPS) was invented to improve on previous search methods for reversible circuits, especially the incompletely specified functions.

The MPS algorithm can synthesize initial functional specifications that are complete and reversible, but is tuned more towards synthesizing incompletely specified functions. This method generates no ancilla bits, as it assumes that ancilla bits have been generated earlier in the MI-MO-IR preprocessing algorithm.

The MPS algorithm uses several cost functions as goals guiding the heuristic search. This is in itself similar to our early variants of the Greedy Search algorithm (not discussed in the dissertation). But the concept of using various cost functions in this section creates gates dynamically (when it runs) rather than taking them from a fixed library. This property is similar in turn to the MMD and MP family of algorithms. This new “Multi-Parameter Search Algorithm” is a result of our analysis of the previous algorithms, their successes or their lack of successes. This algorithm searches from outputs to inputs, until it finds an identity function in the leaf of the branch of the tree. It can keep backtracking to improve the cost of the circuit. Moreover, we can calculate quantum costs of gates rather than other costs, which makes this method feasible for quantum technologies. The costs can be also adapted to other realization technologies of gates.

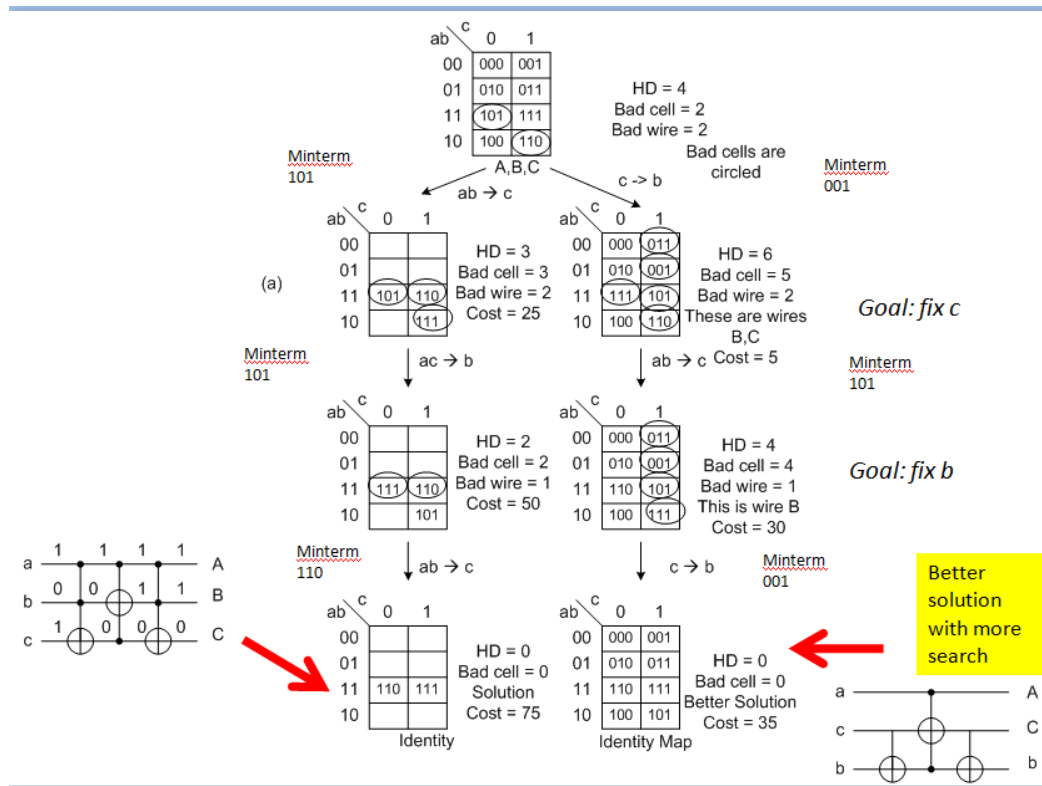
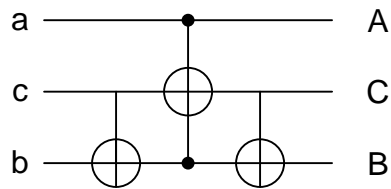


Figure. 6.2.2.1. Synthesis of Fredkin Gate using “Multi-parameter Search Algorithm MPS” using tree search. Solutions found by “multi-parameter algorithm” are given in Figure 6.2.2.3. (left branch of the tree) and Figure 6.2.2.2. (right branch). Notation  $ab \rightarrow c$  means executing  $c = c \oplus ab$  gate. Cells left empty in left branch are self-mapping minterms. This notation emphasizes concentration on greedy maximizing the number of self-mapping minterms. The minterms currently processed are listed. The goals are to fix variables  $c$  and next  $b$  in order to minimize the number of bad wires.

This algorithm is a tree search, guided by four heuristics: Hamming Distance HD, number of bad cells, number of bad wires and the cost of the intermediate solution.

**Example 6.2.2.1.**

Let us realize function from Figure 6.2.2.1, the bad cells for every KMap are encircled. “Bad cells” are minterms that are different than what they should be in the identity Kmap. The numbers of bad wires are also given next to any KMap, “Bad Wires” are qubits (wires) that are different than what they should be in the identity Kmap. In the KMap at right in the third row from top the bad wire is B.



*Figure 6.2.2.2 The solution found by “Multi-parameter Algorithm”. This is the circuit found in the right branch of the tree from previous Figure 6.2.2.1*

MPS “Multi-parameter Search (MPS) Algorithm” finds the well-known circuit of Fredkin Gate from Figure 6.2.2.2 This solution has much lower quantum cost than solution found by MMD and other search algorithms. With good combination of heuristics for bad cells, bad wires, HD and cost, the first branch of the tree finds this solution. With bad choices of weights of cost function more backtracking is necessary.

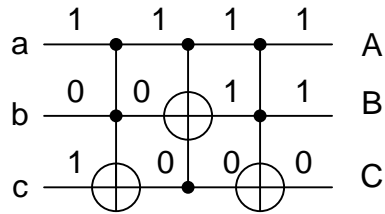


Figure 6.2.2.3 Non-minimal synthesis of the Fredkin Gate. This is the solution found by the MMD algorithm. It has a higher quantum cost than the solution found by the “Multi-parameter Search Algorithm”. This solution is also found in the left branch of the tree from Figure 6.2.2.1. Simulation for input data 101 is shown. This way we can verify results of our algorithms for all input vectors.

To emphasize the role of cost functions, the function from Figure 6.2.2.1 was completely specified as it is easier to differentiate choices with numerical values of partial cost functions than when function is completely specified. Now we will illustrate the new method for functions with don’t cares on two more examples.

### **Example 6.2.2.2**

Given below (in Figure 6.2.2.4.) is an illustration how the algorithm works for an incomplete specification. The tree illustrates how the algorithm finds the minimum cost circuit for this specification.

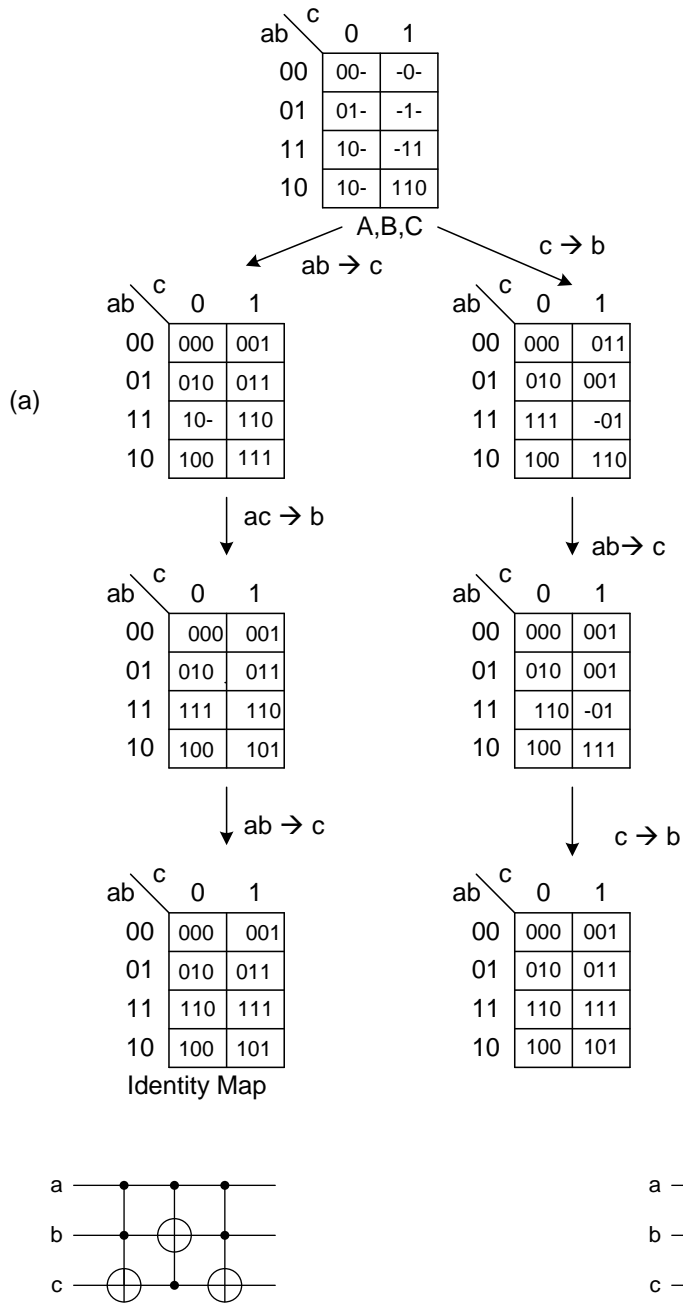
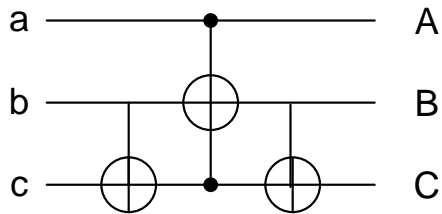


Figure 6.2.2.4. Synthesis of Fredkin Gate using “Multi-parameter Algorithm MPS” and tree search for a function with incomplete specification. Solutions found by “multi-parameter algorithm” are given in Figure 6.2.2.5 (left branch of the tree) and Figure 6.2.2.6 (right branch). Notation  $ab \rightarrow c$  means executing gate  $c = c \oplus ab$ .

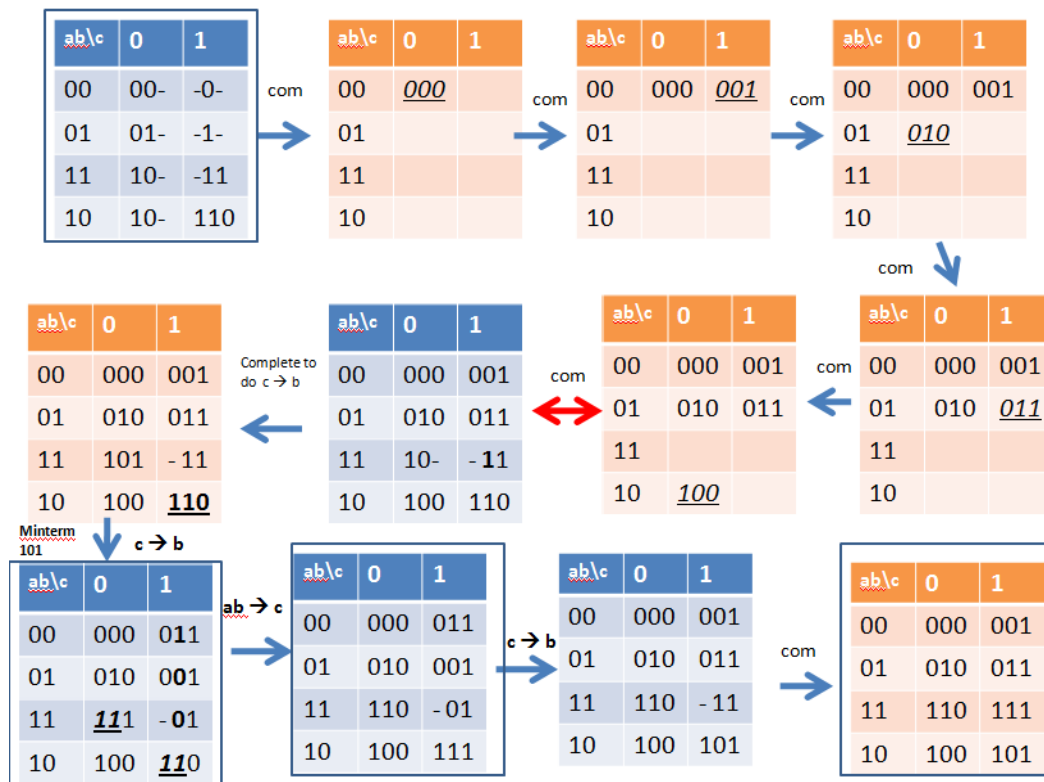
This algorithm illustrated in Figure 6.2.2.4 is a tree search, guided by four heuristics: Hamming Distance HD, number of bad cells, number of bad wires and the cost of the intermediate solution. However, it deals now with don't cares, so it can complete don't cares to cares in various ways. One way is to complete the whole KMap (truth table) in all possible ways using backtracking and next synthesize reversible circuit for each of them. The other way is to complete dynamically don't cares in such a way that simple gates are realized which cover relatively large part of the map, minimizing some or all of our cost functions.



*Figure 6.2.2.5. The solution found by “Multi-parameter Algorithm MPS”. This is the circuit found in the right branch of the tree from previous Figure 6.2.2.4.(this is same circuit as in Fig 6.2.2.2)*

MPS Algorithm for incomplete specification finds in this example the well-known circuit of the Fredkin Gate from Figure 6.2.2.5. It would be not possible for previous algorithms.

A more detailed analysis of the second branch of the tree from Figure 6.2.2.4 is presented in Figure 6.2.2.6. Blue tables are initial table and those that reversible gates are applied to them. Orange tables are intermediate stages based on step-by-step completions of don't cares to cares. By com we denote the relation of completing a “backtrack-completeable” function to a more completely specified reversible function by changing some selected cares to cares. They are selected to allow realization of inexpensive gates and realization of some synthesis goals such as minimizing the number of bad wires or bad cells or other cost function.



*Figure 6.2.2.6. Detailed analysis of operations performed in the right branch of the tree from Figure 6.2.2.4. By symbol com near blue arrows we denote completion of don't cares to cares. By symbol com near read arrow we denote the final backtrack-completeable function obtained as a sequence of don't care completion. This function cannot be further completed to self-mapping minterms so a gate must be selected. The algorithm further converts don't cares to cares – this time with the goal of allowing application of gate  $b = b \oplus c$  (denoted by  $c \rightarrow b$ ). This completion is done to realize minterm 101. However, application of the gate destroys now the previously determined self-mapping minterms, for instance 001. This means that in future the operation should be undone with a mirror. This in reality happens with next operation of  $c \rightarrow b$  operation in the previous to last stage. In the last stage the final don't care is converted to a care in the only possible way to keep the function to be completely reversible. As the result the algorithm obtains a completely specified identity function, the same as all our algorithms.*

In theory, the Multi-parameter Search Algorithm MPS is always able to find the best solution based on backtracking, even for functions with don't cares and that are initially not reversible, but can be completed to reversible without adding ancilla bits. The complex heuristic function which I use in this algorithm for don't cares, reduces the size of the search space.

Below I show the complete backtracking method. It can be used, as shown, for all minterms, or for some subset of minterms. Explanation is easier for a case when the

function is completely converted to correct reversible completely specified data, which is illustrated here.

To explain the new algorithm, it is helps to use either KMaps and trees or succession of truth tables.

In the example below we will use the succession of truth tables.

**Example 6.2.2.3.**

Given is an incomplete specification from Figure 6.2.2.7. let us find the minimum cost reversible cascade for this specification.

Inputs (xyz)	Outputs (PQR)
000	0XX
001	010
010	00X
011	1X1
100	1X1
101	1XX
110	1XX
111	00X

Figure 6.2.2.7. Truth Table of the initial specification for the incompletely specified 3\*3 function

$P, Q, R = F(x, y, z)$  for MPS algorithm.

MPS algorithm converts outputs PQR to completely specified binary values so that F becomes reversible, and then executes the synthesis. We will illustrate these two steps using sequence of truth tables.

In Figure 6.2.2.8, the black color indicates a newly assigned bit combination that is repeated i.e. it appeared previously in the truth table. Gray represents an assignment that is temporarily valid (i.e. it leads to a value that hasn't been used yet, but could turn out later to be invalid as the MPS moves further down the table). Observe that similar to MP algorithm, the output vectors in Figure 6.2.2.8 become transformed from top to bottom and once a vector is transformed, it will not be changed again. This is seen as part of each  $S_i$  column above the grey level row.

S1	S2	S3	S4	S5
0XX	000	000	000	000
010	010	010	010	010
00X	00X	000	001	001
1X1	1X1	1X1	1X1	101
1X1	1X1	1X1	1X1	1X1

1XX	1XX	1XX	1XX	1XX
1XX	1XX	1XX	1XX	1XX
00X	00X	00X	00X	00X

*Figure 6.2.2.8. Output vectors generated by MPSRS in steps S1-S5.*

Column S1 is the outputs (PQR) column from Figure 6.2.2.7. For the first pass of column S1, MPS finds the first output that contains "don't cares." Then assigns binary values to the two "don't care" symbols. It compares this new assignment with the expected assignment and finds that it is valid, as illustrated in column S2 of Figure 6.2.2.8. MPS then continues down the outputs in the truth table until another assignment can be made. When it finds the next output that has "don't cares" it applies the same steps as in S2. Column S3 in Figure 6.2.2.8 indicates the bit combination 000 has already been used.

S6	S7	S8	S9	S10
000	000	000	000	000
010	010	010	010	010
001	001	001	001	001
101	101	101	101	101
101	111	111	111	111
1XX	1XX	100	100	100

1XX	1XX	1XX	100	101
00X	00X	00X	00X	00X

Figure 6.2.2.9. Output vectors generated by MPS in steps S6-S10.

S11	S12	S13	S14	S15
000	000	000	000	001
010	010	010	010	010
001	001	001	000	00X
101	101	101	1X1	1X1
111	111	111	1X1	1X1
100	100	100	1XX	1XX
110	110	110	1XX	1XX
00X	000	001	00X	00X

Figure 6.2.2.10. Output vectors generated by MPS in steps S11 - S15.

Partial assignment S13 proves that one of earlier assignments was invalid as there are no possible assignments to 00X that could lead to unique Input/Output mapping. Thus, the

first instance of backtracking starts at S14. The backtracking to the invalid assignment occurs and the re-assignment is performed as shown in S14 (Figure 6.2.2.10).

When MPS backtracks, it references to the data structure for the original output values before they had been assigned binary values. It then restores the original output values to the last output that had "don't cares." MPS then tries a new combination. If that new combination still results in assignments that are not completely valid, MPS backtracks to an earlier point in the process.

If that does not work, MPS backtracks to another possible invalid assignment as shown in S15 (Figure 6.2.2.10 ).

S16	S17	S18	S19	S20
001	001	001	001	001
010	010	010	010	010
000	000	000	000	000
1X1	101	101	101	101
1X1	1X1	101	111	111
1XX	1XX	1XX	1XX	100

1XX	1XX	1XX	1XX	1XX
00X	00X	00X	00X	00X

Figure 6.2.2.11. Output vectors generated by MPS in steps S16 – S20 with backtracking.

S21	S22	S23	S24
001	001	001	001
010	010	010	010
000	000	000	000
101	101	101	101
111	111	111	111
100	100	100	100
100	101	110	110
00X	00X	00X	000

Figure 6.2.2.12. Output vectors generated by MPS in steps S21 – S24.

S25	S26	S27	S28	S29
001	010	010	011	011
010	010	010	010	010
000	00X	00X	00X	000
101	1X1	1X1	1X1	1X1
111	1X1	1X1	1X1	1X1
100	1XX	1XX	1XX	1XX
110	1XX	1XX	1XX	1XX
001	00X	00X	00X	00X

Figure 6.2.2.13. Output vectors generated by MPS in steps S25-S29 with backtracking.

In S26, MPS backtracks again and re-assigns a value to the first minterm (Figure 6.2.2.13).

S30	S31	S32	S33	S34
011	011	011	011	011
010	010	010	010	010
000	000	000	000	000
101	101	101	101	101
1X1	101	111	111	111

1XX	1XX	1XX	100	100
1XX	1XX	1XX	1XX	100
00X	00X	00X	00X	00X

*Figure 6.2.2.14. Output vectors generated by MPS in steps S30 – S34.*

S35	S36	S37	S38
011	011	011	011
010	010	010	010
000	000	000	000
101	101	101	101
111	111	111	111
100	100	100	100
101	110	110	110
00X	00X	000	001

*Figure 6.2.2.15. Output vectors generated by MPS in steps S35 - S38.*

At the end of stage S38 (Figure 6.2.2.15), a completely specified reversible function  $F(x, y, z)$  is found as shown in Figure 6.2.2.16.

Inputs (xyz)	Original Outputs (PQR)	Final Outputs (PQR)
000	0XX	011
001	010	010
010	00X	000
011	1X1	101
100	1X1	111
101	1XX	100
110	1XX	110
111	00X	001

*Figure 6.2.2.16. Output Original function versus final function  $F(x, y, z)$  after MPS application.*

After obtaining the final truth table (Figure 6.2.2.16), this truth table is then provided as an input data to the MP algorithm to synthesize the reversible cascade. The new completely specified function has a cycle of length 7 and is thus difficult to realize.

Another method of converting the function specification to a completely specified function without backtracking is by creating incomplete KMaps of each function P, Q and R separately, as discussed in section 6.2.1. This type of conversion for our example is shown in Figure 6.2.2.17. The truth table is decomposed to three individual truth

tables, illustrated in Figure 6.2.2.17. It is verified that the function can be converted to a completely specified function without adding ancilla bits. Next ESOP minimization is used the same way as in section 6.2.1. It can be found in the KMap of function P that the only solution to this function is expression  $P = x \oplus yz$ . We denote this signal as P' and we know that it cannot be changed by next stages. However, we can use P' to synthesize other outputs, Q and R. The new truth table is shown in Figure 6.2.2.18.

xy \ z		0	1
		0	1
00		0	0
01		0	1
11		1	0
10		1	1

P' =  
P

xy \ z		0	1
		0	1
00		-	1
01		0	-
11		-	0
10		-	-

Q

xy \ z		0	1
00		-	0
01		-	1
11		-	-
10		1	-

R

Figure 6.2.2.17. The KMaps for  $P$ ,  $Q$  and  $R$  separately obtained from the original incomplete specification in Figure 6.2.2.10.

Inputs (xyz)	New Inputs ( $P'yz$ )	Final Outputs (PQR)
000	000	0XX
001	001	010
010	010	00X
011	011	00X
100	100	1X1
101	101	1XX
110	110	1XX
111	111	1X1

Figure 6.2.2.18. The truth table after applying the first gate from inputs, with output in wire  $P'P$ .

Finally we synthesize  $Q$  and  $R$  as functions of  $P'$ ,  $y$  and  $z$ , as shown in Kmaps from Figure 6.2.2.19.

$P'y \backslash z$		0	1
00		-	1
01		0	0
11		-	-
10		-	-

$Q$

$P'y \backslash z$		0	1
00		-	0
01		-	-
11		-	1
10		1	-

$R$

Figure 6.2.2.19. Realization of output functions  $Q$  and  $R$  as functions of variables  $P'$ ,  $y$  and  $z$ .  $Q = y'$ ,  $R = y' \oplus z$

The final circuit is shown in Figure 6.2.2.20 and the final truth table of complete specification realized by this mapping is shown in Figure 6.2.2.21.

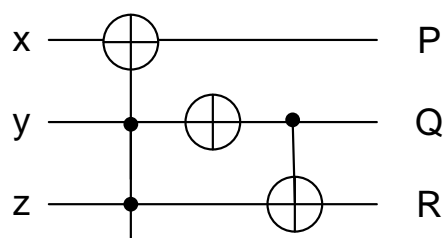


Figure 6.2.2.20 Cascade realized by the second variant of synthesis for Example 6.2.2.3. This method uses the algorithm from section 6.2.1.

Inputs (xyz)	New Inputs ( $P'yz$ )	Final Outputs (PQR)
000	000	011
001	001	010
010	010	000
011	011	001
100	100	111
101	101	110
110	110	100
111	111	101

Figure 6.2.2.21 The final truth table after applying all gates.

Concluding, the MPS is always able to find the best solution based on backtracking. Its heuristic function reduces the search space (size of the tree being searched).

### 6.2.3. EXPERIMENTAL RESULTS FOR THE MULTI-PARAMETER SEARCH ALGORITHM

	3_20	3_40	3_60	3_80	6_20	6_40	6_60	6_80	9_20	9_40	9_60	9_80
orig	16	5	8	5	313	303	303	120	4006	4027	3959	4007
new	16	5	6	5	313	303	303	118	4006	4027	3959	4007

*Table 6.2.2.1. Comparison of results of MPS and DCARL on quantum costs. Orig means DCARL, new means MPS. Names of functions have percent of don't cares as the second part of the name. The first part of the name tells how many qubits the function has.*

The Table 6.2.2.1 above compares results from MPS and DCARL of Manjith Kumar. This comparison was done on the same benchmarks as are used by DCARL. We cannot do comparison on other benchmarks as there are none available. Observe that MMD, Agarwal/Jha and other algorithms were not tested on incomplete functions. As we see in the table the new solutions are better in two cases and the same in all other cases. This is a small improvement of quantum cost of DCARL, the only algorithm to which a comparison may be done. We believe that a better improvement of quantum cost would be possible for functions of more than 9 variables, but this cannot be tested.

The above results testify that a further work on synthesis of incomplete functions is necessary. Not all functionalities discussed above were presented and the algorithm can be further improved. Perhaps better results would be obtained with more iterations of MMDSN orders allowed in search. Other orderings should be also tested and compared.

### **8.3. CONCLUSIONS**

This chapter presented several background ideas important to create efficient synthesis algorithms for reversible cascades from relational, incomplete and reversible specifications:

1. Section 6.1 introduced a new and general method for synthesis of reversible cascades specified by incomplete permutative matrices and arbitrary gates. This method is for  $n \times n$  reversible circuits and creates no ancilla bits. It uses arbitrary gates. This method is easy to program in MATLAB as it is based on matrix calculus.
2. Section 6.2.1 introduced tree search algorithm to synthesize irreversible functions with incomplete specifications. This method

controls how many ancilla bits to use, it synthesizes multiple-input multiple-output reversible or irreversible functions and uses an ESOP minimizer. This method is a pre-processing method for MP-like algorithms. In particular we use MPS from section 6.2.2 for further processing.

3. Section 6.2.2 introduced another tree search method for incomplete specifications. It is applied to only backtrack-completeable functions, which means such functions that can be completed to standard reversible functions without adding ancilla bits. This method (MPS) improved on results from DCARL algorithm.

Although my results are positive (results from MPS outperformed those from DCARL as in table 6.1), I feel that more research in this area of synthesis and better improvements can be made.

## **Chapter 7: CONCLUSIONS**

### **7.1. Conclusions**

In this dissertation, several new theories for synthesis of reversible circuits with no ancilla bits and with small ancilla bits have been developed for completely specified and incompletely specified functions. A software tool MP has been also developed which is currently the best software for reversible circuits synthesis in the sense that it can synthesize large functions and it gets least cost results (on average). Two other software tools were also developed, 2-3-S-A for synthesis with Y gates and MPS for synthesis of incompletely specified functions. The 2-3-S-A tool is the first algorithm ever designed to synthesize with “pseudo-reversible” specifications, and the MPS tool is currently the best available software to synthesize from incomplete specifications.

Using realistic quantum costs, our MP software was shown to be superior to all other software tools for this task, especially for large reversible functions. We can synthesize functions with 32 variables instead of functions with approximately 12 variables or less for previous software tools developed by me and other authors. This way, experimental numerical results confirmed the high quality of the new algorithm. MP is now a top tool in the world, as discussed in ISMVL 2010 conference where several authors of MMD and other reversible synthesis software participated (this conference is the main event in the area of reversible synthesis).

In addition to MP, this dissertation develops an approach to synthesize reversible circuits using a new type of gate, the Y gate. This  $2 \times 3$  gate has been not used for reversible logic synthesis so far. Technically it is not a reversible gate, and at the advice of participants of ISMVL 2010 conference we will call it a pseudo-reversible gate. The circuits built with these gates can have unequal number of inputs and outputs, so they do not conform to the classical definition of reversible circuits – we call them pseudo-reversible circuits for now and we are open to change the name, however it is important to emphasize that these gates are a new concept to logic synthesis, so if the Y-gate technology becomes successful, logic synthesis with pseudo-reversible gates will become a popular research subject such as synthesis with reversible gates became a popular subject 10 years ago. My algorithm works only for 3 or 4 bit circuits, but it is the first algorithm and software tool ever created for synthesis of pseudo-reversible circuits. There are some other pseudo-reversible gates that we know now and each of them can become a potential research subject for logic synthesis algorithms.

The table below summarizes the goals of this dissertation, new theories developed and how the goals were achieved experimentally.

**Table 7.1 Goals of the dissertation and theories developed**

Goal	Specific theories and algorithms in chapters	How this task was experimentally achieved?
<p><b><u>Goal 1:</u></b> (size of functions). Being able to synthesize reversible cascades (circuits) for reversible completely specified functions of <u>such large size that there exists no any tool available to synthesize them</u>. The size of the function is in terms of the number of variables and the number of Generalized Toffoli gates in corresponding circuit.</p>	<p><b><u>Chapter 5.</u></b> <i>New algorithm –MP generalizes and improves the MMD algorithm. Generalized ordering of minterms. Minterms are implicit and not explicit thus a new realization is created. Version for completely specified multi-output functions. It has two versions of ordering, MMDS and MMDSN, the last one for large circuits.</i></p>	<p>1. <i>MMDS algorithm was created and tested. It was found to have good performance but not able to run on functions larger than MMD.</i></p> <p>2. <i>MMDS was modified to MP in order to run on larger functions.</i></p> <p>3. <i>MP was tested and proven to run on very large functions that MMD and MMDS are not able to synthesize.</i></p> <p>4. <i>I was thus able to show that not only I can have better results (in term of cost) on known</i></p>

		<i>benchmarks but that my software can synthesize functions that are so large that none of the previous approaches were able synthesize.</i>
<p><b><u>Goal 2:</u></b> (don't Cares) Being able to synthesize reversible circuits for functions that have don't cares in their initial specifications. The costs of the synthesized circuits should be smaller than the cost obtained using the only available existing tool DCARL.</p>	<p><b><u>Chapter 6.</u></b> <i>New approaches to synthesis of reversible cascades for incomplete specification by adding ancilla bits has been created. Variants were also created that do not add ancilla bits.</i></p> <p><i>The theoretical study of several variants for solving these tasks have been created.</i></p>	<ol style="list-style-type: none"> <li><i>1. I implemented algorithm MPS from Chapter 6.</i></li> <li><i>2. I tested the MPS algorithm and compared its operation on benchmarks.</i></li> <li><i>3. The results were on average better than for DCARL.</i></li> </ol>

<p><b><u>Goal 3:</u></b> (<i>reduced costs</i>)</p> <p>Being able to synthesize reversible circuits which have smaller costs than the costs of those produced by the top tool, MMD.</p>	<p><i>Theory and algorithms to reduce costs were developed in <b><u>Chapters 5 and 6.</u></b></i></p>	<p><i>This goal is reached by MP program.</i></p>
<p><b><u>Goal 4:</u></b> (<i>various cost functions</i>) Being able to synthesize reversible functions with circuit cost values for several technologies, such as quantum or optical, and not just the number of gates or literal costs used now.</p>	<p><b><u>Chapters 3 and 4</u></b> discuss <i>cost functions for various technologies. A uniform synthesis method realizes Toffoli family of gates in each technology and next synthesizes with these gates.</i></p>	<p><i>Program MP uses gate cost and quantum cost to evaluate solutions.</i></p>
<p><b><u>Goal 5:</u></b> (<i>new technologies</i>)</p> <p>Being able to synthesize reversible circuits for new types of reversible logic, for which no algorithm was ever</p>	<p><i>This goal is reached in <b><u>Chapters 3 and 4.</u></b> I discuss various technologies in which <math>k \times k</math> Toffoli gates can be</i></p>	<p><i>Software tool 2-3-S-A was created that finds exact minimum solutions to small reversible functions.</i></p>

<p>created. This new technology of Y-switches has gates with different numbers of inputs and outputs, the 2-3 switch has 2 inputs and 3 outputs.</p>	<p><i>built that are next used in all kinds of algorithms. Some other gates used in algorithms from chapters 5-6 are also presented.</i></p> <p><b><u>Chapter 4.</u></b> <i>New types of reversible gates are given and methodology of synthesis is presented.</i></p>	
<p><b><u>Goal 6: (relational)</u></b> Being able to synthesize reversible functions for relational specifications, which are more general than “functions with don’t cares” and which occur in synthesis of reversible circuits and reversible automata.</p>	<p><i>Theory and algorithm were developed in <b><u>Chapter 6</u></b> that allow to synthesize from relational specification in a form of an incomplete permutative matrix of a reversible function.</i></p>	<p><i>This algorithm was not programmed.</i></p>
<p><b><u>Goal 7: (irreversible)</u></b> Being able to synthesize a reversible</p>	<p><i>Theory and algorithms were developed in</i></p>	<p><i>Other algorithms were not programmed.</i></p>

circuit from a functional specification that is not reversible.	<b><u>Chapter 6</u></b> <i>to synthesize from irreversible specifications.</i>	<i>MPS allows for a limited conversion</i>
---	--	--

I initially considered synthesizing functions with algorithms that would run faster than the MMD. This task was however considered less important, as discussed with my PhD committee during the proposal defense. I tested on many benchmarks my algorithms for speed: MMDS Algorithm, Greedy algorithm (new) and MP Algorithm for this task. Some examples run faster but on average I was not able to reach this goal and I resigned from achieving it. It seems that it is not possible to have a program that is both higher quality, runs on larger data and faster at the same time. All other goals of my dissertation have been achieved.

I tested MP on larger and more diversified sets of benchmarks than Dmitri Maslov or any other previous author. The goal of such exhaustive benchmarking was to find for what types of problems which tools are better, which can help myself and next authors to improve them. It is for instance known that random functions are the most difficult in case of standard binary logic, but the reversible functions seem to be all in this most difficult category. I have compared my results mostly against results of MMD software, as the comparisons of MMD versus Agrawal/Jha and other software have been already published in the literature.

## References

- [AgrawalJha04] A. Agrawal and N. K. Jha. "Synthesis of reversible logic," in Proc. DATE, Paris, France, pp. 710- 722, February 2004.
- [AgrawalJha04b] A. Agrawal, and N. K. Jha, Synthesis of reversible logic. In Proceedings Design and Test in Europe Conference, Paris, France, February 2004, 1530-1591, or 1384-1385.
- [Akashdeep05] A. Akashdeep, report, PSU, 2005.
- [Alhagi10] N. Alhagi, M. Hawash, and M. Perkowski, Synthesis of Reversible Circuits with No Ancilla Bits for Large Reversible Functions Specified with Bit Equations Proc. 40th IEEE International Symposium on Multiple-Valued Logic ISMVL. Barcelona, Spain , May 26-May 28, ISBN: 978-0-7695-4024-5
- [AlRabadi01] A. Al-Rabadi, and M. A. Perkowski, "Multiple-Valued Galois Field S/D Trees for GFSOP Minimization and Their Complexity". Proc. ISMVL 2001, pp. 159-166.
- [AlRabadi01a] A. Al-Rabadi, and M. Perkowski "On the Characterization of Reversible Multi-Valued Galois Logic Primitives" Proc. LDL'2001 Symposium, PSU, Portland, Oregon, 29-30 August 2001
- [AlRabadi01b] A. Al-Rabadi, and M. Perkowski "New Classes of Multi-Valued Reversible Decompositions for Three-Dimensional Layout", Proc. Reed-Muller'2001, pp. 185-204, Starkville, Mississippi, 10-11 August 2001.
- [AlRabadi02a] A. Al-Rabadi, L. Casperson, M. Perkowski and X. Song, "Multiple-Valued Quantum Logic," Proc. ULSI 2002, Boston, May 15, 2002.
- [AlRabadi02b] A. Al-Rabadi, L. Casperson, and M. Perkowski "Canonical Forms and Synthesis for Multiple-Valued Quantum Logic", Proc. ULSI 2002.
- [Athas] W. C. Athas, and L."J." Svensson , "Reversible Logic Issues in Adiabatic CMOS", Exploratory Design Group, University of Southern California - Information Sciences Institute, Marina del Rey, CA 90292-6695, {athas,svensson}@isi.edu
- [Barenco95] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and W. H., "Elementary gates for quantum computation," The American Physical Society, no. 5, pp. 3457–3467, 1995.
- [Bennett73] C. H. Bennett, "Logical Reversibility of Computation", IBM Journal of Research and Development, 17, 1973, pp. 525-532.
- [Bennett82] C. H. Bennett. The thermodynamics of computation – a review. IJTP, 21(12): 905—940, 1982.
- [Bennett89] C.H. Bennett. Time/space trade-offs for reversible computation. SIAM Journal on Computing, pages 766 – 776, 1989.

- [Biamonte05b] J. Biamonte, M. Jeong, J. Lee, M. Perkowski, "Extending Classical Test to Quantum," Proceedings of SPIE "Fluctuations and Noise in Photonics and Quantum Optics, Editors: P.R. Hemmer, J.R. Gea-Banacloche, P. Heszler, Sr., M. S. Zubairy, Vol. 5842, pp. 194-205, May (2005), doi: 10.1117/12.623715. III.
- [Biamonte07] J. Biamonte and M. Perkowski, "A Quantum Test Algorithm," Submitted to IEEE Transactions on Computers and quant-ph/0501108.
- [Bratko01] I. Bratko. PROLOG, Programming for Artificial Intelligence. 2001. 3<sup>rd</sup> edition
- [Brayton89] R. Brayton and F. Somenzi, An Exact Minimizer for Boolean Relations, Proc. Int. Conf. on Computer Aided Design ICCAD, 1989.
- [Bryant86] R. E. Bryant. Graph-Based Algorithms for Boolean function Manipulation. *IEEE Trans. Computers*, C-35(8), pp. 677-691, 1986.
- [Chrzanowska97] M. Chrzanowska-Jeske, Z. Wang, Y. Xu, "A Regular Representation for Mapping to Fine-Grain, Locally-Connected FPGAs," *Proc. Intern. Symp. On Circuits and Systems ISCAS'97*, pp. 2749-2752, June 1997.
- [Chrzanowska99] M. Chrzanowska-Jeske, Y. Xu, M. Perkowski, "Logic Synthesis for a Regular Layout," *VLSI Design*, Vol. 10, No. 1, pp. 35 - 55, 1999.
- [Cuykendall87] R. Cuykendall and D. McMillin, "Control-Specific Optical Fredkin Circuits," *Applied Optics*, 26, pp. 1959-1963, 1987.
- [DeBenedici07] De Benedici, Report on future technologies for supercomputing, SANDIA Laboratories, 2007.
- [DeVos00] A. De Vos, "Design of reversible logic circuits by means of control gates", Proceedings Patmos 2000 Conference, Goettingen, (Springer Lecture Notes in Computer Science No. 1918) pp. 255-264, 15 September 2000.
- [DeVos01] A. De Vos, "Control gates as building blocks for reversible computers", Proc. Patmos 2001 Conference, Yverdon. paper 9.2. 28 September 2001.
- [DeVos02] A. De Vos, B. Raa, and L. Storme, "Generating the Group of Reversible Logic Gates", *Journal of Physics A: Mathematical and General*, vol. 35, 2002, pp. 7063-7078.
- [Dill97] K. M. Dill, "Growing Digital Circuits: Logic Synthesis and Minimization with Genetic Operators", M. S. Thesis, Department of Electrical and Computer Engineering, Oregon State University, June 1997.
- [Dill97a] K. M. Dill and M. A. Perkowski, "Minimization of Generalized Reed-Muller Forms with a Genetic Algorithm", *Proc. of Genetic Programming '97*, July 1997, Stanford University, California.
- [Donald08] J. Donald and N.K. Jha, Reversible Logic Synthesis with Fredkin and Peres Gates, *ACM Journal on Emerging Technologies in Computing Systems*, Vol. 4, No. 1, Article 2, March 2008.

- [Dueck03] G. W. Dueck and D. Maslov, "Reversible Function Synthesis with Minimum Garbage Outputs," 6th International Symposium on Representations and Methodology of Future Computing Technologies (RM), Trier, Germany, March 2003, pp. 154-161.
- [Dueck03b] G. W. Dueck, D. Maslov, and D. M. Miller, "Transformation-based synthesis of networks of Toffoli/Fredkin gates," in Proc. IEEE Canadian Conf. Electrical and Computer Engineering, May 2003, pp. 211–214.
- [Dueck03c] G. Dueck, D. Maslov, and D. M. Miller, "A Transformation Based Algorithm for Reversible Logic Synthesis," Proc. DAC. 2003 Anaheim, CA, pp. 318-323
- [Elspas71] S. Elspas, "The Theory of Multirail Cascades," in "Recent Developments in Switching Theory," (ed. A.Mukhopadhyay), Academic Press, New York and London, 1971, pp. 315 – 367.
- [Forsberg04] E. Forsberg, Reversible logic based on electron waveguide Y-branch switches, Nanotechnology 15, pp. 298– pp. 302, 2004.
- [Forsberg05] E. Forsberg, The Electron Waveguide Y-Branch Switch: A Review and Arguments for its Use. Preprint.
- [Fredkin82] E. Fredkin and T. Toffoli, "Conservative logic", Intern. J. Th. Physics, 21, pp. 219-253, 1982.
- [Giesecke06] N. Giesecke: Ternary Quantum Logic. M.S. thesis, PSU, Dept ECE, 2006.
- [Gilchrist03] A. Gilchrist, G.J. Milburn, W.J. Munro, and K. Nemoto, "Generating optical nonlinearity using trapped atoms," Hewlett-Packard. 2003
- [Grosse06] D. Große, X. Chen, and R. Drechsler, Exact Toffoli network synthesis of reversible logic using boolean satisfiability, Proc. IEEE Dallas/CAS Workshop, pp. 51–54, 2006.
- [Grosse09] D. Große, R. Wille, G.W. Dueck, and R. Drechsler. Exact multiple control toffoli network synthesis with SAT techniques. IEEE Trans. on CAD, 28(5):703–715, 2009.
- [Grover96] L. Grover, "A fast quantum mechanical algorithm for database search," Proceedings of the 28th Annual ACM Symposium on Theory of Computing 1996, pp. 212-219 1996. [Online].
- [Gupta06] P. Gupta, A. Agrawal, and N. Jha. An algorithm for synthesis of reversible logic circuits. IEEE Trans. on CAD, 25(11):pp. 2317–2330, 2006.
- [Hawash10] Maher Hawash, PhD dissertation in preparation, PSU 2010.
- [Hein05] J. L. Hein. Prolog Experiments in Discrete Mathematics, Logic, and Computability. 2005.<http://web.cecs.pdx.edu/~payel/CS251/PrologLabManual.pdf>

- [Hossain09] S. Hossain, Ph.D. Dissertation, ECE Dept, January 2009.
- [Hu06] J. Hu G-S Ma, G. Feng, Efficient Algorithm for Positive-polarity Reed-muller Expansions of reversible circuits, Proc. 18th Intern. Conf. on Microelectronics (ICM) 2006, pp. 63-66.
- [Jha06] Jha. "An Algorithm for Synthesis of Reversible Logic Circuits." IEEE Trans. Computer-Aided Design, 2006.
- [Kerntopf01] P. Kerntopf. A Comparison of Logical Efficiency of Reversible and Conventional Gates. Proc. IWLS'01.
- [Kerntopf04b] P. Kerntopf. "A new heuristic algorithm for reversible logic synthesis," in Proc. DAC, pp. 834-837, June 2004.
- [Khan04a] M. H. A. Khan and M. A. Perkowski, "Genetic Algorithm Based Synthesis of Multi-Output Ternary Functions Using Quantum Cascade of Generalized Ternary Gates", Proc. 2004 Congress on Evolutionary Computation, Portland, OR, USA, 19-23 June 2004, pp. 2194-2201.
- [Khan06] Khan F., Perkowski M.: Synthesis of Hybrid and d-Valued Quantum Logic Circuits by Decomposition. Theoretical Computer Science. Vol. 367, Issue 3, 2006, pp. 336-346.
- [Khlopoutine02] A. Khlopoutine, M. Perkowski, and P. Kerntopf, "Reversible logic synthesis by gate composition," *Proceedings of IWLS 2002*, pp. 261 – 266.
- [Kumar07] M. Kumar, B. Year, N. Metzger, Y. Wang, and M. Perkowski, "Realization of Incompletely Specified Functions in Minimized Reversible Circuits," Proc. RM 2007.
- [Kumar08] Realization of Incompletely Specified Reversible Functions MS. Thesis. PSU, 2008.
- [Landauer61] R. Landauer, "Irreversibility and Heat Generation in the Computational Process", IBM Journal of Research and Development, 5, 1961, pp. 183-191.
- [Lee06] S. Lee, S. J. Lee, T. Kim, J-S. Lee, J. Biamonte, and M. Perkowski, "The Cost of Quantum Gate Primitives," Journal of Multi-valued Logic and Soft Computing, Vol. 12, No. 5-6. 2006.
- [Lee97] G. Lee. Logic synthesis for cellular architecture FPGA using BDD. Proc. ASP-DAC, pp. 253-258, 1997.
- [Lee99] G. Lee, R. Drechsler, and M. Perkowski. ETDD-Based Synthesis of Two-Dimensional Cellular Arrays for Multi-Output Incompletely Specified Boolean Functions," IEE Proc. Comput. Digit. Tech, Vol. 146, No. 6, November 1999, pp. 302 - 308.
- [Lukac02] M. Lukac, M. Pivtoraiko, A. Mishchenko, and M. Perkowski, "Automated Synthesis of Generalized Reversible Cascades using Genetic Algorithms", Proc. Fifth

Intern. Workshop on Boolean Problems, pp. 33-45, September 19-20 2002, Freiberg, Sachsen, Germany.

[Lukac02a] M. Lukac, and M. Perkowski, "Evolving Quantum Circuits Using Genetic Algorithms", Proc. of 5th NASA/DOD Workshop on Evolvable Hardware 2002, pp. 177- 185.

[Lukac03] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, Ch-. H. Yu, K.Chung, H. Jee, B. Kim, Y.D. Kim, "Evolutionary Approach to Quantum and Reversible Circuits Synthesis", Artificial Intelligence Review Journal, Special Issue on Artificial Intelligence in Logic Design.

[Lukac05] M. Lukac, M. Perkowski, "Combining Evolutionary and Exhaustive Search to Find the Least Expensive Quantum Circuits", Proceedings of the 14th International Workshop on Post-Binary ULSI Systems, May 18, 2005, Calgary, Canada.

[Lukac05a] M. Lukac and M. Perkowski, "Using exhaustive search for the discovery of new families of optimum universal permutative binary quantum gates," Proc. International Workshop on Logic and Synthesis, June 2005.

[Lukac09] M. Lukac, Ph.D. Dissertation, Electrical Engineering Department, PSU, January 2009.

[Maher10] H. Maher, Ph.D. Dissertation, Electrical Engineering Department, PSU, 2010 in preparation.

[Margolus87] N. Margolus, Physics and Computation, MIT PhD Thesis (1987). Reprinted as Tech. Rep. MIT/LCS/TR415, MIT Lab. for Computer Science, Cambridge MA.

[Margolus03] N. Margolus, Universal cellular automata based on the collision of soft spheres. New construction in Cellular Automata, Oxford Press, 2003.

[Maslov03] D. Maslov and G. Dueck, "Improved quantum cost for k-bit Toffoli gates," IEE Electron. Lett., vol. 39, no. 25, pp. 1790–1791, Dec. 2003.

[Maslov03a] D. Maslov and G. W. Dueck, "Garbage in reversible design of multiple output functions," in Proc. 6th Int. Symp. Representations and Methodology of Future Computing Technologies, Mar. 2003, pp. 162–170.

[Maslov03b] D. Maslov, "Reversible Logic Synthesis", PHD dissertation, 2003.

[Maslov04] D. Maslov and G. W. Dueck, "Reversible cascades with minimal garbage," IEEE Transactions on CAD, 23(11): pp.497-1509, November 2004.

[Maslov07] D. Maslov, G.W. Dueck, and D.M. Miller, Techniques for the Synthesis of Reversible Toffoli Networks, ACM Transactions on Design Automation of Electronic Systems, Vol. 12, No. 4, Article 42, Sept. 2007.

[Maslov08] D. Maslov, G.W. Dueck, D. M. Miller, and C. Negrevergne, Quantum Circuit Simplification and Level Compaction, IEEE Trans. on CAD, Vol. 27, No. 3, March 2008, pp 436-444.

- [Maslov10] D. Maslov, Web Page: <http://webhome.cs.uvic.ca/~dmaslov/>
- [Maslov05c] D. Maslov and M. Miller, Reversible and Quantum Circuit Simplification: Circuit  $\rightarrow$  Circuit, and D. Maslov and M. Miller, Methods for Reversible Logic Synthesis: Creating A Circuit, Proc. LDL 2005.
- [Meurers01] D. Meurers. Implementing finite state machines and learning Prolog along the way. 2001.<http://www.sfs.unituebingen.de/~dm/07/winter/684.01/slides/03.pdf>
- [Miller03] D. M. Miller and G.W. Dueck, "Spectral Techniques for Reversible Logic Synthesis," Proc. RM 2003, pp. 56-62.
- [Miller03a] D. M. Miller, D. Maslov and G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis", Proc. Design Automation Conference, Anaheim, pp. 318–323, June 2003.
- [Miller05] D. Miller, and E. Fredkin, Two- state, Reversible, Universal Cellular Automata in Three Dimensions. Proceedings of the 2nd Conference on Computing Frontiers, Ischia, Italy, pp. 45—51, 2005.
- [Miller05b] M. Miller and D. Maslov, Finding a Set of Optimal 3-line NCV Circuits, Proceedings of LDL 2005.
- [Miller08] M. Miller, Quantum Multi-Valued Decision Diagrams, 2008, ISMVL.
- [Mishchenko01] A. Mishchenko and M. Perkowski. "Fast Heuristic Minimization of Exclusive Sum-of-Products," Proc. Reed-Muller Workshop, 2001, Starkville, Mississippi, pp. 242-250.
- [Mishchenko02] A. Mishchenko and M. Perkowski, "Logic Synthesis of Reversible Wave Cascades", Proc. IEEE/ACM International Workshop on Logic Synthesis, June 4-7, 2002, pp. 197 – 202.
- [Mishchenko08] A. Mishchenko, "EXTRA Library of the DD procedures," <http://www.ee.pdx.edu/~alanmi/research/extra.htm>
- [Mohammadi08] M. Mohammadi, and M. Eshaghi, Heuristic methods to use don't cares in automated design of reversible and quantum logic circuits. Quantum Inf. Process. J. 7(4), pp. 175–192, 2008.
- [Muthukrishnan00] A. Muthukrishnan, and C. R. Stroud, Jr., "Multivalued logic gates for quantum computation", Physical Review A, Vol. 62, No. 5, Nov. 2000, 052309/1-8.
- [Nielsen00] M. Nielsen and I. Chuang, "Quantum Computation and Quantum Information," Cambridge University Press, 2000.
- [Patino10] A. Patino, MSc Thesis in preparation, Department of Electrical Engineering, PSU, 2010.

[Peres85] A. Peres, "Reversible Logic and Quantum Computers," *Physical Review A*, 32:3266-3276, 1985.

[Perkowski93] M. Perkowski, E. Pierzchala. New Canonical Forms for Four-Valued Logic. *Technical Report*, Electrical Engineering Department, PSU, 1993.

[Perkowski97] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. S. Zhang, "Decomposition of Multiple-Valued Relations", *Proc. of the International Symposium on Multi-Valued Logic* 1997, St. Francis Xavier University, Antigonish, Nova Scotia, Canada, May 28-30, 1997, (Piscataway, New Jersey: IEEE 1997).

[Perkowski97e] M. Perkowski, M. Chrzanowska-Jeske, Y. Xu, "Lattice Diagrams Using Reed-Muller Logic," *Proc. RM'97 Conference*, Oxford Univ., U.K., Sept. 1997, pp. 85 - 102.

[Perkowski97f] M. Perkowski, E. Pierzchala, R. Drechsler, "Layout-Driven Synthesis for Submicron Technology: Mapping Expansions to Regular Lattices," *Proc. First International Conference on Information, Communications and Signal Processing, ICICS'97*, Singapore, 9-12 Sept. 1997.

[Perkowski01] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, Alan Coppola, B. Massey, "Regularity and Symmetry as a Base for Efficient Realization of Reversible Logic Circuits", *Proceedings of IWLS 2001*, pp. 90 - 95.

[Perkowski01a] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, A. Coppola, B. Massey, "Regular realization of symmetric functions using reversible logic," *Proceedings of EUROMICRO Symposium on Digital Systems Design*, 2001, pp. 245-252.

[Perkowski01a] M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al-Rabadi, A. Coppola, A. Buller, X. Song, M. M. H. A. Khan, S. Yanushkevich, V. Shmerko, and M. Chrzanowska-Jeske, "A general decomposition for reversible logic," *Proceedings of RM 2001*. pp. 119 – 138.

[Perkowski01b] M. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Mishchenko, and M. Chrzanowska-Jeske "Three-Dimensional Realization of Multiple-Valued Functions using Reversible Logic", *Proc. of the 10th International Workshop on Post-Binary ULSI Systems '2001*, pp. 47-53, Warsaw, Poland, 21 May 2001.

[Perkowski02] M. Perkowski, A. Al-Rabadi, and P. Kerntopf, "Multiple-Valued Quantum Logic Synthesis," *Proc. of 2002 International Symposium on New Paradigm VLSI Computing*, Sendai, Japan, December 12-14, 2002, pp. 41-47.

[Perkowski03] M. Perkowski, M. Lukac, M. Pivtoraiko, P. Kerntopf, M. Folgheraiter, D. Lee, H. Kim, H. Kim, W. Hwangboo, J.-W. Kim, and Y.W. Choi, "A Hierarchical Approach to Computer Aided Design of Quantum Circuits," *Proceedings of 6th International Symposium on Representations and Methodology of*

*Future Computing Technology*, RM 2003, Trier, Germany, March 10-11, 2003, pp. 201-20X

[Perkowski10] M. Perkowski, N. Alhagi, M. Lukac, N. Saxena, and S. Blakley, *Synthesis of Small Reversible and Pseudo-Reversible Circuits Using Y-Gates and Inverse Y-Gates*, ISMVL 2010, on CD

[Picton94] P. Picton, "Modified Fredkin Gates in Logic Design," *Microelectronics Journal*, **25**, 1994, pp. 437-441.

[Picton00] P. Picton. A Universal Architecture for Multiple-valued Reversible Logic. *MVL Journal*, 5, 2000, pp.27-37.

[Priese76] L. Priese, "On a Simple Combinatorial Structure Sufficient for Sublyling Nontrivial Self-Reproduction," *J. Cybernet.* 6, 101, 1976.

[Rice09] J. E. Rice, K. B. Fazel, M.A. Thornton, and K. B. Kent, Toffoli Gate Cascade Generation Using ESOP Minimization and QMDD-based Swapping. *Proc. RM*, May 23-24, 2009, pp. 63-72.

[Riley06] P. T. Riley. The Temple of Quantum Computing. 2006. Version 1.1. [http://www.toqc.com/TOQCv1\\_1.pdf](http://www.toqc.com/TOQCv1_1.pdf)

[Rubinstein01] B.I.P. Rubinstein, "Evolving quantum circuits using genetic programming", *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001)*, pp. 144-151, 2001.

[Saeedi07] M. Saeedi, M. Sedighi, M. S. Zamani, A Novel Synthesis Algorithm for Reversible Circuits, *Proc. ICCAD*, San Jose, California, pp. 65-68. 2007.

[Saeedi07a] M. Saeedi, M. Sedighi, M. Saheb Zamani, A New Methodology for Quantum Circuit Synthesis: CNOT-Based Circuits as an Example," *Proc. IWLS*, 2007. Paradise Point Resort and Spa, San Diego, CA, USA, May 30-June 1.

[Saeedi07b] M. Saeedi, M. S. Zamani, M. Sedighi, On the Behavior of Substitution-Based Reversible Circuit Synthesis Algorithms: Investigation and Improvement," *Proc. ISVLSI*, 9-11 March 2007, pp. 428-436.

[Sarabi94] A. Sarabi, N. Song, M. Chrzanowska-Jeske, M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *Proc. DAC'94*, San Diego, June 1994, pp. 321 - 326.

[Sasao01] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic and Synthesis (IWLS01)*, Lake Tahoe, CA, June 12-15, 2001. pp. 225-230.

[Sasao78] T. Sasao, K. Kinoshita, "Cascade realization of 3-input 3-Output Conservative Logic Circuits", *IEEE Trans. on Computers*, **27**, 1978, pp. 214-221.

[Sasao79] T. Sasao, K. Kinoshita, "Conservative Logic Elements and Their Universality," *IEEE Trans. on Computers*, 28, 1979, pp. 682-685.

[Sasao74] T. Sasao, K. Kinoshita, "A Catalog of Magnetic Bubble Logical Circuits for Three-Variable Logical Functions," *Technology Reports of the Osaka University*, 24, No.1169, pp. 133-140, 1974.

[Shamir86] J. Shamir, H. J. Caulfield, W. Micelli, and R. Seymour, "Optical Computing and the Fredkin Gates," *Appl. Opt.* 25, 1604, 1986.

[Saxena09] N. Saxena and S. Blakely, Report, PSU, March 2009.

[Shah10] Shah Dipal, Ph.D. in preparation, 2010. PSU.

[Shende02] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Reversible logic circuit synthesis", in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, pp. 125-132, November 10-14, 2002.

[Song93a] N. Song, M. A. Perkowski, "A New Design Methodology for Two-Dimensional Logic Arrays," *Proc. of IEEE International Workshop on Logic Synthesis, IWLS '93*, Tahoe City, CA, pp. 1 - 17, May 1993.

[Song95] N. Song, M. Perkowski, M. Chrzanowska-Jeske, and A. Sarabi, "A New Design Methodology for Two-Dimensional Logic Arrays," *VLSI Design*, 1995, Vol. 3., Nos. 3-4, pp. 315-332.

[Song98] N. Song and M. Perkowski, "Minimization of Exclusive Sums of Multi-Valued Complex Terms for Logic Cell Arrays," *Proc. ISMVL 98*, Fukuoka, Japan, May 1998.

[Spector99] L. Spector, H. Barnum, H.J. Bernstein, and N. Swamy, "Finding a better-than-classical quantum AND/OR algorithm using genetic programming," *Proc. 1999 Congress on Evolutionary Computation*, Vol. 3, pp. 2239-2246, Washington DC, 6-9 July 1999, IEEE, Piscataway, NJ.

[Tarau07] P. Tarau and B. Luderman. A Logic Programming Framework for Combinational Circuit Synthesis. Report.

[Toffoli80a] T. Toffoli. "Reversible computing," Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci, 1980.

[Toffoli80b] T. Toffoli. "Reversible Computing," In *Automata, Languages and Programming*, Springer Verlag, 1980, pp. 632- 644.

[Welzer] T. Welzer, I. Rozman, and J. Gyorkos. Using Prolog for Digital Circuits Simulation. Report.

- [Wille08] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. Proc. 38<sup>th</sup> ISMVL, pp. 220–225. Dallas, TX, USA, 2008. <http://www.revlib.org>.
- [Wille09] R. Wille, R. Drechsler, BDD-based Synthesis of Reversible Logic for Large Functions. Proc. DAC, pp. 270-275. San Francisco, California
- [Wille09a] R. Wille, M. Saeedi, R. Drechsler, Synthesis of Reversible Functions Beyond Gate Count and Quantum Cost. IWLS 2009.
- [Viamontes08] G. Viamontes, QuIDDPro Webpage, [vlsicad.eecs.umich.edu/Quantum/qp/req.html](http://vlsicad.eecs.umich.edu/Quantum/qp/req.html)
- [Yang04] G. Yang, W. Hung, X. Song, and M. Perkowski, “Exact synthesis of 3-qubit quantum circuits from non-binary quantum gates using multiple-valued logic and group theory,” International Journal of Electronics, 2004.
- [Yang05] G. Yang, W. N. N. Hung, X. Song and M. Perkowski, "Exact Synthesis of 3-qubit Quantum Circuits from Non-binary Quantum Gates Using Multiple-Valued Logic", IEEE/ACM Design Automation and Test in Europe (DATE), Munich, Germany, March 2005.
- [Yang05h] G. Yang, X. Song, M.A. Perkowski, W.N.N. Hung, and J. Biamonte, “The Power of Large Pulse-Optimized Quantum Libraries: Every 3-qubit Reversible Function can be Realized with at Most Four Levels,” Proc. International Workshop on Logic and Synthesis, June 2005.
- [Yen04] B. Yen, N. Denler, M. Perkowski, “Synthesis of Ternary Logic Using Generalized Ternary Gate Cascades in a Filtering Model Approach,” Proc. ULSI 2004.
- [Younis94] S.G. Younis, “Asymptotically Zero Energy Computing Using Split-Level Charge Recovery Logic, Ph.D. Thesis, MIT, June 1994

## 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.

#### **Boolean relation**

Initial specification that has some kind of constraints on input and output values that are more general than in traditional don't cares. For instance, a Boolean relation may specify that input 0011 is mapped to output 0000 or output 1111, but not to other binary output vectors. Boolean relations cannot be specified by using only don't cares, but they are generalizations of the don't care concept.

## 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.

### **Conservative function**

Conservative Boolean function is a Boolean function that has the same number or “ones” in each pair of input minterm to output minterm mapping, for instance in mapping 00110  $\rightarrow$  11000 both the input minterms (binary vector) and the output vector have 2 ones.

### **Controlled-V**

Controlled-V and Controlled- $V^+$  are controlled gates that control truly quantum single-qubit gates called “Square root of NOT” and “Square-root-of-NOT-hermitian”, respectively. V is a gate such that  $V*V = \text{NOT}$ ,  $V^+ * V = I$  (identity). These gates are used internally in quantum to build (quantum) Toffoli gate from 2\*2 quantum primitives.

### **Convergent logic synthesis algorithm**

Convergent algorithm is an algorithm that guarantees to synthesize a correct solution (a circuit that meets the specification). The solution does not have to be minimal or even quasi-minimal. It is enough if it is correct.

## 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.

### **DCARL**

DCARL is the first software tool for synthesis of incompletely specified functions to reversible circuits. It was designed by Manjith Kumar at PSU.

### **Double-Rail logic**

Double-rail logic is a method to build circuits in which every logic value is represented by two physical signals, for instance variable  $a$  is represented by wire  $a$  and wire  $a'$ . In some technologies universal reversible gates can be realized only in double-rail logic.

## **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.

### **Exact synthesis.**

Research area that produces synthesis methods (usually exhaustive search algorithms) that synthesis produces results for which it can be proven that the cost function reaches the minimum. This is a rather theoretical area that has little application in practical industrial software CAD tools.

## **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.

### **Feynman gate**

Reversible gate invented by Feynman. It has two inputs and two outputs and repeats one variable on the first output and realizes EXOR of both input variables on the second output. This gate is used to realize linear and affine (with NOT gate) reversible circuits. It

is inexpensive so the algorithms should increase the number of such gates and reduce Toffoli gates and other gates with more than 2 bits.

### **Fredkin gate**

Reversible gate invented by Fredkin. It is also called Controlled SWAP. When the top control input is one the gate swaps two bottom bits. Otherwise nothing happens and gate is a logical identity.

## **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). Garbages waste energy in non-quantum 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.

### **Generalized reversible gates**

Generalized reversible gates are gates that extend some well-known gate, controlled by one or two qubits, to a gate controlled by many qubits. For instance, the Toffoli gate can be generalized from  $A=a, B=b, C=ab \oplus c$  to  $A=a, B=b, C=c, D=d, F=abcd \oplus f$ . Toffoli is a “generalized Feynman gate”. There are also generalized Fredkin gates and other.

### **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.

### **Hybrid Gates**

Hybrid gates have some of their bits realized with binary and other with multiple-valued data. Realization of hybrid quantum gates is relatively easy, in contrast to CMOS or traditional technologies.

## **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**

### **Kerntopf gates**

Gates invented by Kerntopf that have the maximum numbers of cofactors among all reversible gates with 3 or 4 bits. They are thus good for synthesis of circuits with ancilla bits as setting some inputs to constants we can create many functions with Kerntopf gates.

### **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 their teams since 2003.

### **MP algorithm**

Software to minimize reversible circuits written in the framework of this dissertation.

### **Multiple-Valued reversible logic**

Multiple-Valued reversible logic realizes circuits using multiple-valued gates. Such gates have more than two values. For instance, ternary gates have three logic values; 0, 1 and 2.

## **N**

### **NMR computers**

Quantum computers that use Nuclear Magnetic Resonance (NMR) to realize quantum circuits. The circuit is virtual, realized in time by changing “quantum spins” of “nuclear particles”. The 2010 technology allows to build 10-qubit NMR computers, which have only experimental value but are a proof of technology.

## **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.

### **Pseudo-Kronecker Functional Decision Diagrams (PKFDDs).**

Diagrams that extend KFDDs by allowing to have several expansion types in each level (for each variable). They are generalizations of KFDDs.

## 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.

### **Quantum dot**

Technology to realize circuits. There are standard quantum dots, reversible quantum dots and truly quantum “quantum dots”. The first realizes Boolean functions, the second reversible circuits and the third types realize truly quantum circuits (not necessarily reversible but also those with superposition and entanglement).

### **Quantum multiplexer**

Multiplexer-like circuit realized in quantum circuits where the values of the control bits select the quantum operators performed in the data path. Quantum multiplexers are described by special types of unitary matrices and synthesis methods are developed for them.

### **Quantum Oracle.**

Quantum Oracle is an oracle realized in a quantum permutative circuit. Oracle is a Boolean circuit with many inputs and one output that answers yes/no to certain question formulated by values of input variables. An oracle is also called a predicate or a concept in various branches of computer science.

### **Quantum register**

Quantum register is an ordered vector of qubits. The name comes from a “register” of “quantum particles” such as ions which are kept in order by external forces. Quantum gates are realized by operations on one or two qubits from the quantum register.

## **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**

### **Scaleable synthesis algorithm**

Algorithm that is able to synthesize larger circuits in a reasonable time, one for which the time and space complexities allow to solve problems of practical size.

### **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  $U^+$  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.

## **Y**

### **Y gate**

Y-gate is a common nano gate, also called the “Priese Switch” or the “2-by-3 switch”. It is used in fluidic, optical and other new technologies.