1-2009

Classical Search and Quantum Search Algorithms for Synthesis of Quantum Circuits and Optimization of Quantum Oracles

Sazzad Hossain
Portland State University

CLASSICAL SEARCH AND QUANTUM SEARCH ALGORITHMS FOR SYNTHESIS OF QUANTUM CIRCUITS AND OPTIMIZATION OF QUANTUM ORACLES

by

SAZZAD HOSSAIN

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

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
2009

DISSERTATION APPROVAL

The abstract and dissertation of Sazzad Hossain for the Doctor of Philosophy in Electrical and Computer Engineering were presented January 14, 2009, and accepted by the dissertation committee and the doctoral program.

COMMITTEE APROVALS:

[redacted]

Marek Perkowski, Chair

[redacted]

James Morris

[redacted]

Xiaoyu Song

[redacted]

Douglas Hall

[redacted]

Charles Weber
Representative of the Office of Graduate Studies

DOCTORAL PROGRAM APPROVAL:

[redacted]

Malgorzata Chrzanowska-Jeske , Director
Electrical and Computer Engineering
Ph.D. Program

ABSTRACT

An abstract of the dissertation of Sazzad Hossain for the Doctor of Philosophy in Electrical and Computer Engineering presented January 14, 2009.

Title: Classical Search and Quantum Search Algorithms for Synthesis of Quantum Circuits and Optimization of Quantum Oracles.

We observe an enormous increase in the computational power of digital computers. This was due to the revolution in manufacturing processes and controlling semiconductor structures on submicron scale, ultimately leading to the control of individual atoms. Eventually, the classical electric circuits encountered the barrier of quantum mechanics and its effects. However, the laws of quantum mechanics can be also used to produce computational devices that lead to extraordinary speed increases over classical computers. Thus quantum computing becomes a very promising and attractive research area. The Computer Aided Design for Quantum circuits becomes an essential ingredient for such emerging research which may lead to these powerful computers to be realized—an era of Quantum computing. This thesis presents an integrated theoretical study of software algorithms to design circuits of quantum oracles as well as methods for designing quantum oracles for Grover algorithm to solve combinatorial problems. An implementation of quantum algorithm involves the

initialization of the input state and its manipulation with quantum gates followed by the measurements. In Grover algorithm the problem to be solved is specified by a permutative logic oracle – the fundamental problem is then how to build this oracle from quantum logic circuits and how to optimize these circuits. These problems are NP-hard and require search algorithms. In future, the search will be also done in quantum and this thesis leads to quantum algorithms to design quantum circuits more efficiently.

DEDICATION

This thesis is dedicated to:

My parents;

My wife;

My daughters;

ACKNOWLEDGMENTS

I am deeply grateful to my dissertation advisor, Professor Marek Perkowski, who allowed me the opportunity to take the next step in my scientific vocation and do my PhD. He allowed me the scope to follow my thoughts and develop my scientific skills, whilst always being available to provide guidance, inspiration and discussion whenever it was needed over a period of several years. I would like to thank Professor Xiaoyu Song and Professor Fu Li for serving on both of my dissertation proposal defense and comprehensive exam committee. I would like to extend special thanks to Professor James Morris, Professor Xiaoyu Song, Professor Douglas Hall and Professor Charles Weber for being on my dissertation committee. Special thanks to Professor James Morris for stimulating discussion and sharing with me his incredible breadth of knowledge in Quantum Mechanics and nanotechnology. Again, special thanks to Professor Charles Weber for his invaluable review of my dissertation presentation. Special thanks to the Governing authority of International Islamic University Chittagong, Bangladesh for their constant inspiration and financial support. I would like to thank my friends for their support and inspiration. I also gratefully acknowledge my friends Shamsul Abedin and his wife Morsheda Khatun for their continuous inspiration and support. Lastly but not leastly, I would like to express my gratitude to my parents, my wife Leila Hossain, daughters Sanzida Hossain and Tanzila Hossain for their love, understanding, patience and support sustained me through the end of my PhD.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

Introduction

1.1. Why Quantum Computers are superior to classical Computers

This thesis is devoted to some aspects of designing quantum computers. One may ask "Why quantum computers are of interest and why are they more powerful than standard computers realized in CMOS technology?"

1. First, a quantum computer operates on a qubit (quantum bit) and not bit. Much more information can be contained in a qubit than in a bit. While a bit has only one bit of information, 0 or 1, the qubit can be represented by a point on a sphere. So, theoretically qubit has an infinite capacity (this sphere is called a Bloch Sphere [Nielsen00]). However, the information in the qubit is so-called "hidden" which means that to know this information some special processing must be executed and some of this information will be lost. If we measure the qubit in the simplest way, it is probabilistically converted to a normal bit, thus we measure the value of 0 or 1, with certain probabilities. A memory of qubits can store much more information than a standard memory.

2. Second, normal logic gate can be in one state, 0 or 1. But a quantum gate can be in any superposition of states $|0\rangle$ and $|1\rangle$ which are quantum states (basis states) corresponding to 0 and 1, respectively. These are just two points on the

Bloch Sphere. Superposition is of the form $\alpha|0\rangle + \beta|1\rangle$ where α and β are complex numbers called quantum amplitudes. These values are so constrained that they correspond to all points on the surface of the sphere. It can be showed that this is equivalent to $|\alpha|^2 + |\beta|^2 = 1$. This means that a quantum circuit calculates in parallel on many values, and the scale of this parallelism is orders of magnitude higher than in any classical parallel system available now or ever. This is called quantum parallelism.

3. When measured, a bit collapses to $|0\rangle$ with probability $|\alpha|^2$ and to $|1\rangle$ with probability $|\beta|^2$. Thus a probabilistic computer can be easily simulated on a quantum computer. We know that a (classical) probabilistic computer is more powerful than a deterministic (classical) computer. Based on the above a quantum computer is at least as powerful as a classical deterministic and probabilistic computer. It is however much more powerful, but for not all problems.

4. There is one more source of power of quantum computers, the most important one. It is called entanglement and it results from the fact that quantum amplitudes are complex numbers. The entanglement is the resource that exists only in quantum mechanics. It does not exist in classical physics and is very difficult to simulate in it. It is treated now by physicists as a fundamental

resource of the Universe, on par with matter, energy and information. Now only three types of computing are known: deterministic, probabilistic and entangled and, only quantum computer has them all. The entanglement is a constraint on states that the quantum system can have. When we add or multiply classical probabilities we never get a value zero. However adding and multiplying quantum amplitudes zero can be created which means that some states are excluded. This is the fundament of creating quantum states being solutions in many quantum algorithms.

1.2. Towards Quantum CAD

1.2.1. The idea of using a quantum computer to design a quantum computer

It is popularly known, even among non-specialists that modern computers and all integrated circuits are built using computer using Computer Aided Design software. Humans are just not able to deal with enormous complexity of such designs without the use of computers in all stages of designing, optimizing, verifying, validating and testing modern systems.

It is however less well known that several basic problems in Computer Aided Design

of standard logic circuits are NP hard. This means that they are optimization problems that are counterparts of NP complete decision problems. NP complete problems allow verifying that S is a solution to problem P in polynomial time but they need exponential time to find the solution. The solution is of Yes/No type. An example of such problem is Satisfiability where we have a Boolean formula and we have to answer a question: *"does there exist an assignment of values to Boolean variables from the formula such that this formula is satisfied?"*

Classical circuits are designed using AND, OR and NOT gates (logic or Boolean operators). We call it the AND/OR base. Quantum computers are designed with AND, EXOR and NOT base and they are reversible (chapter 2). The synthesis and optimization problems in quantum computing are even more difficult than classical problems as in the classical reversible logic there is no possibility to find a general structure like AND/OR or no general decomposition of a large problem to smaller ones. Standard AND/EXOR logic methods cannot be used without modifications.

The field of synthesis of binary (classical) reversible logic is very new. The synthesis problems become even more difficult when the CAD system is built to synthesize and minimize quantum circuits. The contemporary algorithms for this task are based mostly on heuristic and evolutionary ideas, or are based on matrix algebra but applicable only to very small quantum circuit specifications.

The reason of these difficulties is a much more complex model of a quantum circuits. For instance, Richard Feynman observed that quantum mechanics problems are very difficult to solve on a classical computer. This observation caused him to conclude – "we need a quantum computer to model quantum mechanical phenomena efficiently". While working on test of quantum circuits, Biamonte and Perkowski [Biamonte04, Biamonte05, Biamonte05a, Biamonte05b] observed that testing of quantum circuits is much simpler when quantum phenomena themselves are applied. Therefore the following idea may came to mind – "May be similar bootstrapping can exist in the area of synthesizing quantum circuits? May be a quantum computer can allow to solve problems efficiently for which the standard computer is very inefficient". This thesis, among other new ideas, tries to answer this question and realize this intuition by designing conceptual quantum circuits, blocks and oracles that will become useful for automated synthesis with the introduction of practical quantum computers.

But before quantum computers will become available we still have to design them using classical computers. So the thesis is also interested in how to use standard computers to design quantum computers efficiently. We will show relations between these two problems of classical and quantum design of quantum circuits.

1.2.2. Quantum Computer Aided Design Using Grover Algorithm

Grover algorithm to search unstructured data base is perhaps the most important and practical quantum algorithm. In this research a new approach to solve several hard problems of Computer Aided Design, and particularly logic synthesis of quantum oracles for Grover algorithm is given. CAD of quantum circuits is one of the most important prerequisites to build a practical quantum computer. Grover algorithm speeds up all NP problems quadratically. There are thousands of such problems, many of them of high practical use, especially in CAD of classical circuits.

We assume here a hypothetical, yet to be build quantum computer, and analyze what would be its use in the area of Computer Aided Design of quantum computers. It has to be pointed out that although we speculate on the existence of a quantum computer with tens of thousands of qubits, we do not speculate on the physical reality of all quantum phenomena such as quantum parallelism, superposition and entanglement, since all these amazing phenomena have been already verified experimentally [Bennett93, Cleve98, Knill05, Nielsen98]. For instance the Grover and Shor algorithms have been already experimentally verified independently in several quantum techonologies [Chuang95, Chuang98, Grover98]. Thus our situation can be compared to that of George Boole and Charles Babbage speculating about the power of computers based on mechanical switches and Boolean algebra in year 1850 – the theory exists and the experimental base exists, but more theory is needed to build

practical circuits and more theoretical/experimental/development work is necessary to develop adequate technology for practical use. Quantum CAD will happen because humanity never goes back on existing scientific and technological possibilities. The problem is only who will build the fundament for this new research area of "*Quantum CAD*" and when.

1.3. Solving problems by reducing them to basic combinatorial search problems

Many generic combinatorial problems are known in classical logic synthesis such as satisfiability, graph coloring, binate covering, spectral transforms based on butterflies and others. Many of these problems are known as Constraint Satisfaction Problems where some solution must be found that satisfies a set of constraints such as equalities and inequalities. In many problems the solution must additionally optimize certain cost function (such as energy). Spectral transforms are another wide class of problems with many applications, just to mention the ubiquitous Fast Fourier Transform or the Fast Cosine Transform used in MPEG.

We will demonstrate in this thesis that these and other problems still remain a fundament for efficiently designing quantum computers. How then to solve these

problems on a quantum computer? How to use a quantum computer to calculate spectra of Boolean functions? How to use a quantum computer to minimize reversible circuits and reversible automata?

Can be this done in principle? Nobody is surprised now that a standard computer can minimize its own circuits better than any human on the earth, but the author of this thesis knows from his Ph.D. advisor, Dr. Perkowski that when he was a beginning engineer the top authorities believed that only a human can optimize logic circuits. The author of this thesis is deeply convinced that future quantum computers will be able to solve problems that are absolutely out of reach not only for a human, but also for the whole Earth supercomputers of 2007 connected by the Internet. These will be not only isolated and abstract problems like factoring large numbers [Shor94] but the real life problems in weather prediction, global economics, designing new drugs or designing quantum computers. We believe that all these problems can be reduced to some finite set of problems for which FPGA-like quantum hardware will be built. FPGA stands for Field Programmable Gate Array.

What is a standard (binary logic) FPGA? This is a new technology in which the user can program not only the memory as in standard computer, but can program also gates, blocks and their connections using special hardware design languages (such as Verilog or VHDL) and synthesis (CAD) software. This way the digital designer can practically

"build his own computer" for given task programmed by him. FPGAs have truly revolutionized digital design since 1986 and are used in all practical products from simple controllers to supercomputers. We introduce in this thesis a model of FPGA-like parallel quantum computer.

Concluding, our model of a quantum computer proposed in this thesis is a multi-purpose, possibly parallel, programmable, reconfigurable, quantum accelerator connected to a standard computer.

1. Our model is multi-purpose because it is not for arbitrary problem but for only some classes of problems. These classes include however many problems.
2. Our model is parallel because we have available in our system not just a single quantum computer but a collection of computers that share information.
3. Our model is programmable in an analogous way as FPGAs are programmable in modern VLSI technology. FPGA is hardware programmable in classical CMOS technology and our Quantum FPGA is hardware programmable in quantum technology (regardless of its technical details).
4. Our model is reconfigurable in the same way as FPGA systems are reconfigurable in modern system design. This means that the top-level structure of the system can be reconfigured dynamically to another system. Thus a vision processor can be modified to a DSP processor or a sorter. When the basic structure of a quantum reprogrammable hardware is created, it can be

reprogrammed very quickly to arbitrary given application. The existing technology already allows for this but is not yet scaleable.

5. Our general model is quantum as some of the processors (except of the master/programmer processor) in the parallel system are quantum.
6. It is an accelerator to emphasize that only some problems are accelerated, not all problems. Nobody would ever need a quantum computer for word processing, standard laptop will suffice. It is however very likely that future computer games will be accelerated on a quantum computer, as it is a perfect hardware to simulate any kind of physics (rendering, shading, motion, biology).

1.4. Problems in synthesis of quantum circuits

Our thesis statement here is that Quantum computers themselves will be used to optimize and synthesize quantum circuits. It will be the same way as the standard computers are used now to synthesize classical circuits from VHDL specifications (VHDL is Very High Level Design automation Language to specify hardware for VLSI and FPGAs). To aid in inventing these new algorithms a new generalized unified approach is created and investigated in my thesis.

This new approach should be of interest to the quantum logic synthesis community because of its analogies and extensions to that of the classical Reed-Muller Logic and

classical reversible logic. Reed-Muller logic is a specialized spectral approach of logic structures invented by Zhegalkin, Reed and Muller that uses AND and EXOR gates as its base. Classical Reversible Logic was invented by Feynman, Toffoli and Fredkin [Feynman82, Feynman96, Fredkin82]. Classical reversible logic research uses gates such as the Fredkin Gate, Toffoli Gate and Feynman gate. These gates are very different from the gates used in classical logic. Recent research in reversible logic research area can be found in [Lukac02, Lukac02a, Lukac05, AlRabadi02, Khan01, Mischenko02, Khlopotine02, Negotevic02, Dill97b, Perkowski02, Yang05]. An important advantages of reversible circuits are low power and high testability. Biamonte and Perkowski extended the classical test generation algorithm for Reed-Muller logic circuits, created by Reddy, to quantum circuits, using an equivalent of Positive Polarity Reed-Muller logic (PPRM). PPRM is a one type of Reed-Muller Logic, the simplest one. Biamonte and Perkowski showed that by applying superposed test vectors to a quantum circuit instead of using standard tests, the testing time can be dramatically reduced. It is however known that Sarabi and Perkowski [Perkowski95, Perkowski99d, Sarabi99, Chang99], Sasao [Sasao91a], Falkowski [Falkowski03], Bhattacharya [Bhattacharya1] and others generalized the results of Reddy [Reddy72] to even more complex structures than PPRM, still having high testability. The same ideas can be thus perhaps used to quantum circuits. First one has however to find the quantum counterparts of such circuits, which amazingly was not done yet in the literature. In this thesis it will be shown how KRM, FPRM, GRM, ESOP and other

canonical forms and equation types can be extended and generalized to highly testable quantum circuits. Because of the wide scope of the thesis the testability issues themselves, analyzed by new students in PSU Quantum Group, will be not discussed here.

The complexity of synthesizing large circuits of reversible and quantum types exceeds much the complexity of designing classical circuits. Efficient methods for synthesizing them are therefore necessary. The researches of previous Ph.D. students at PSU (Anas Al-Rabadi [Al-Rabadi02], Karen Dill [Dill01], Bruce Yen [Yen05], Martin Lukac [Lukac08]) as well as other researcher's world-wide (Maslov [Maslov05], Viamontes [Viamontes04]) have been only partially successful and there are still no CAD tools for most important problems in quantum circuit synthesis. This thesis is competing with these previous approaches in the sense that we are building here practical CAD tools and using existing computer software technology, for the synthesis of quantum circuits.

Moreover, we speculate also on the future Computer Aided Design tools for this class of problems that will become possible with the availability of quantum computers. One can thus say that this thesis tries to do for both classical and quantum CAD an equivalent of what was done by Peter Shor for computer security, (as related to RSA cracking). It means, enable a new technology. However, while Shor invented a new

quantum algorithm for this purpose, we use the Grover algorithm.

We aim also that our synthesis methods will be not for theoretical specifications only (like reversible truth tables) but for practical data that may appear in designing practical oracles. These specifications are irreversible and hierarchical thus allowing to specify a general class of Grover oracles for large problems and not only for toy problems. Our methodology outlined on many examples in this thesis can be used by other researchers for their own problems if these problems are reducible to Grover algorithm.

1.5. New General-Purpose Search Approaches for classes of combinatorial problems

We will present here the development of a general-purpose quantum search/synthesis/learning meta-algorithm Quantum Search Problem Solver (QSPS) to be used in solving highly complex CAD combinatorial problems, especially the Constraint Satisfaction Problems. QSPS is a new "meta-algorithm" and a general constructive learning methodology. It is applicable for both logic synthesis and minimization. QSPS is designed for logic circuits implemented in quantum hardware, as well as for software applications of logic synthesis such as Data Mining, Machine Learning, off-line Evolvable Hardware, and Knowledge Discovery in Databases

(KDD). Our approach is therefore very general. Although there are other meta-algorithms, QSPS is original and more general than other algorithms of this type.

The QSPS method is based on the general concept of search in certain space of solutions and candidates for solutions. Our search approach has a variant for classical search and another variant for parallel quantum search. The classical search is of course only a special case of the parallel quantum search. Both the classical and the quantum algorithm variants presented here can be improved by this author of other authors in the future. These future improvements will be however based on the main ideas of algorithms presented here. Why I believe this point? Looking to history of classical software search on standard computers it was possible to find new better search methods long time after the concept of search was invented. For instance, the Iterative Deepening Search that I use in Chapter 7 of this dissertation was invented many years after the classical depth-first search and A* search were created. Similarly in the area of quantum search new variants of Grover algorithm have been recently invented by both Grover himself and other authors that are, in one or another way, better than the original Grover Algorithm for some specific problem sub-domains.

Most of our search ideas in quantum case are based on the Grover algorithm [Grover96], one of two most important quantum algorithms known so far. However, we not only use Grover as it is, but we wrap it around in a more general search system

of parallel reconfigurable computers. This system uses parallelism of programming, parallelism of execution, heuristics, reprogramming (as in FPGAs) and it calls Grover Algorithm for sub-problems, possibly with oracles that are adapted and modified. We call this the "dynamic approach to quantum problem-solving based on Grover". By solving some class of problems in Grover, we can learn some parameters to improve the speedup of the next calls of the "Grover Processor". For instance, when one knows the chromatic number of the graph, the optimal coloring of this graph can be found more efficiently by reducing the size of the oracle. Reducing the oracle's size leads to the reduction of the solution time of Grover Algorithm, as will be discussed. Any additional knowledge available to the system designer should be thus used in (parallel) quantum computing to improve the search efficiency. Observe that this is exactly the same problem-solving philosophy as it is used in contemporary standard parallel search algorithms.

Several applications of this quantum meta-learning algorithm will be presented in detail in chapters 12 – 16 of this dissertation, including graph coloring, satisfiability, maximum cliques, SOP minimization, ESOP minimization and others. One application is the minimization of incompletely specified data with FPRM (fixed polarity Reed-Muller forms in which every variable has the same polarity, negated or not negated consistently in the expression). Another application of our approach is the minimization of the GRM (Generalized Reed-Muller) forms (mixed polarities of

variables). FPRMs and GRMs are two most well-known types of the AND/EXOR expressions [Sasao93e]. As the quantum variant of this algorithm is not realizable now, because the largest quantum computers are only for 10 qubits, we may only analyze its simulated behavior and its predicted behavior on future quantum computers, comparing results of quantum algorithm with classical algorithms running on current computers.

We created also a classical model of our quantum search algorithm (special case of QSPS) and we investigate it on the same class of problems. This leads to our Extended Cybernetic Problem Solver (ECPS). ECPS is a simplified and non-quantum model of sequential software search. It is still an efficient tool to solve some CAD problems.

This completes the outline of the basic innovative ideas of my thesis, but there are other new concepts presented below.

1.6. Organization of the thesis with respect to new ideas in logic design

This thesis introduces new ideas in quantum logic design, quantum circuit structures and respective synthesis algorithms and also new ideas in quantum algorithm design. In a sense, "everything relates to everything" in this thesis: "we build

quantum algorithms using quantum circuits to automatically design the next and more improved class of quantum circuits". This multi-aspect core of the thesis makes the presentation difficult. Therefore we organizationally separate the thesis to parts that are relatively less connected. We need also some small text repetitions to simplify the reading of the thesis.

The areas of logic design and algorithm design are respectively isolated, and they are linked by the fact that to build a practical quantum oracle one has to be able to optimize it from quantum gates. For the simplification of introduction, in this chapter we will separate these two ideas in the general presentation. We will link these two ideas more in next chapters when all background will be already introduced. We can thus say that the thesis has two parts, the first part (chapters 2 – 11) relates to the quantum circuit design, the second part (chapters 12 – 15) relates to the quantum oracles design for Grover. Of course, as oracles are built from circuits, the reader has to be familiar with circuit design to understand the oracle design. The first part introduces also some important quantum algorithms including the Grover Algorithm as the core of the thesis. Section 1.6 of this introductory chapter is related to quantum circuits and in section 1.7 we will introduce quantum algorithms, mainly the Grover algorithm.

1.6.1. New circuit structures for permutative quantum logic

The organization of the chapters will be now outlined in more detail. Initially, it will be shown that logical forms for new families of algebras can be developed that are a good match with quantum hardware. These families of forms are analogous to the classical (Reed-Muller) AND-EXOR forms. These families are like the building blocks for reversible circuits. The descriptions of these circuits allow also for an easy conversion of non-reversible specifications to reversible circuits that are realized in quantum. At the beginning, we will show in detail how the proposed basic gates are practically realizable in NMR and we mention briefly also other quantum implementation technologies. A complete logical hierarchy of expansions, trees, decision diagrams, and forms for this new family will be developed to be used in oracles and other quantum (permutative) circuits. These ideas are influenced on one hand by certain algebraic structures, both by those already used in quantum mechanics and by structures used in other technologies. On the other hand, our algebraic structures are influenced by the possibilities of real quantum technology such as those presented in papers of Brassard [Brassard04], Muthukrishan and Stroud [Muthukrishan00] and others. The new concept invented in this thesis is that of "Affine gates" (chapter 7). As "Affine gate" I will define a classical quantum gate such as Toffoli Gate or Controlled-V Gate, which is controlled by an arbitrary affine function. Since affine function is very cheap in quantum realization (as it includes only inexpensive inverters and inexpensive Feynman gates – see chapter 2 for costs of

gates), we use new affine gates instead of classical quantum gates. Our new approach can always lead to the improvements of circuit's cost and speed if a respective synthesis algorithm is implemented. This property results from the fact that the set of the new gates is a superset of the known gates. We can thus talk about "affine Toffoli gate", "affine Fredkin gate" and "affine Controlled-Square-Root-of-Not gate".

The introduced by us logic can also be implemented with hypothetical AND and EXOR gates and is a "regular logic". It means, it has a regular structure of gates and connections, similarly to the well-known classical programmable Logic Arrays (PLAs). Our logic is thus a fundament of building quantum arrays with generalized and affine Toffoli gates. (Quantum array is another name used by specialists for quantum circuits). Our affine AND/EXOR regular reversible array concept is somehow similar to classical PLA, but adapted to reversible quantum circuits. The high degree of quantum testability [Biamonte04], (which generalizes the classical testability concepts [McCluskey97]) for several expression types within the new families of the logics introduced here, provides further motivation for the introduction and study of affine AND/EXOR circuits.

The new family of logic was invented by me from the insights gained by the analogy of the different algebras introduced here (based on literature) and used to minimize logic circuits. Previous research has shown that the hierarchy of Reed-Muller

Expansions can be generalized to a Zhegalkin subset of the Linearly Independent Hierarchy of expansions [Perkowski97a, Perkowski97b, Perkowski97c]. Previous PhD students from PSU, Bogdan Falkowski, Ingo Schafer, X. Zeng, Karen Dill, Ugur Kalay and Anas Al-Rabadi have demonstrated how some decision diagrams and forms [Zeng95] can be obtained by extending the Reed-Muller logic concepts [Dill97a, Dill97b, Dill98, Dill01]. In this dissertation, however, I am able to show some further improvements and generalizations, and also stronger links to modern quantum technology, because some breakthroughs occurred in quantum realizations just very recently, in years 2004-2005 [Biamonte04]. Here we introduce the Quantum Zhegalkin Hierarchy. We use the name Zhegalkin not because he contributed to this logic, but because the name Reed-Muller is already reserved. We propose to keep the name "Reed-Muller circuits" for the RM family binary circuits and to introduce the name "Zhegalkin circuits" to all their counterpart quantum circuits, in order to honor the Russian researcher Zhegalkin [Zhegalkin29] who is unfairly not respected in Western World despite the fact that everybody acknowledges his priority (year 1927) over Irving S. Reed [Reed54] and D.E. Muller [Muller54] (year 1954).

1.6.2. The role of AND-EXOR structured forms in quantum circuit synthesis

One of our approaches to using quantum computers for CAD problems considers the fact that the structured forms, such as FPRMs, are easier to optimize than the

completely unstructured designs such as the general purpose reversible circuits from arbitrary gates. Therefore the quantum approach for FPRM minimization from [Li06] was generalized here to Generalized Reed-Muller (GRM) forms (We developed also classical search algorithm for GRM and compared it to previous research based on the GA Algorithm [Koza99, Dill98]). This is for the first time that the approach to the minimization of Generalized Reed-Muller forms with the quantum algorithm has been attempted. The GRM equation type is a general, canonical expression of the Exclusive-Or Sum-of-Products (ESOPs) type, in which for every subset of input variables there exists not more than one term with arbitrary polarities of all variables. The general-purpose AND-EXOR implementation has been shown in classical CMOS technology to be economical. Generally it requires fewer gates and connections than the AND-OR logic implementations of the same functions. GRM logic is also very highly testable, making it desirable for building permutative quantum oracles (like those used in Grover Algorithm). Most importantly, GRM logic is imminently practical for quantum arrays; this type of logic expression is immediately realizable in quantum hardware and the implementation can be directly compared with that of other algorithms for reversible logic. Our synthesis method converts an arbitrary non-reversible function to a reversible function as the byproduct of the design method itself. The only potential drawback (found only for rare circuits) is an increased number of ancilla qubits. Ancilla qubits are additional qubits added to the circuit to allow the realization of a function that specified this circuit. The previous research

(Dill [Dill97a], Sasao [Sasao93e], Debnath [Debnath95]) has shown the GRM equations are very difficult to minimize, especially with many don't cares. To date, the one exact minimization algorithm developed has required exhaustive searches and is extremely time consuming [Sasao94]. The approximate algorithms [Zeng95, Debnath95, Debnath96, Debnath98] are faster and allow the minimization of larger, completely specified functions. It is however difficult to evaluate the quality of the circuits produced by these algorithms. The goal of using the Quantum Search Problem Solving for GRM minimization is to create exact exhaustive and non-exact heuristic minimization techniques that will produce a higher quality of optimization, i.e. minimized circuits with fewer terms and literals, than that of other heuristic GRM minimization methods. (chapter 8 on GRM ECPS and chapter 15 on FPRM/Quantum). Concluding, the GRM was selected as one example of many canonical forms of Zhegalkin hierarchy, for which some high quality solutions are known [Debnath95, Debnath96, Debnath98] and can be thus compared with.

In the application of minimizing GRM forms, the CGRMIN Software [Software1] was utilized to create a GRM expression for a disjoint input/output table (benchmark). Following this approach, a ECPS algorithm is implemented and Quantum Search Problem-Solving algorithm is described in this dissertation. An interesting property of this search is that only the search space of all "correct", functionally equivalent equations is searched, with the singular task of finding the best reduction. This is in

contrast to all previous algorithms. With this limited-size search space the solutions have absolute guaranteed function coverage. There is no application-specific knowledge incorporated into the method. As such, the results are particularly remarkable since they compare favorably with that of the heuristic algorithms developed by top human experts over several years [Debnath95, Debnath96] in the past. This composition/minimization technique, utilizing the GRM form for the specification of both specified and strongly unspecified functions, by its very nature, is applicable to not only hardware circuit design, but also to the off-line Evolvable Hardware and Data Mining. The methods like this, based on building oracles with complex regular structures in them (in particular butterflies, but also SAT-like circuits), are applicable to many CAD problem formulations.

It has been shown both by this research and other authors [Dill97, Dill98, Dill013, Miller97] that in the logic synthesis process the exhaustive search approaches find circuit implementations that are often different in appearance from those that a human designer would produce. In the outlined logic minimization process, in contrast to many known logic synthesis approaches, the human-designed, application specific heuristics are not the main mechanism to search for solutions. "Could non-human, quantum heuristics be found?" - it remains to be investigated. This thesis does not answer this question. For now we observe experimentally that humans operate only in small subsets of the entire solution spaces. Our software discovered new gates which

were not found by humans and which confirms this observation.

1.6.3. New concepts of synthesis algorithms for particular structures

The application of the benefits gained with the development of the above mentioned new algebras and structures as well as classical circuits, reversible circuits, and standard binary quantum circuits demands that practical, general synthesis and minimization algorithms (CAD tools) be created for them. This general class of quantum algorithms should have a wide applicability to logic problems, for automated logic circuit synthesis and optimization, machine learning, and directed knowledge discovery, because as it will be shown in the sequel, all these research areas are closely related. Although some of these links are known in classical research, they are new in the quantum domain.

It is also desired that good partial heuristic methods be developed for the synthesis algorithm, or rather, class of algorithms. This is thus the second task to be achieved in this thesis. For instance, let us assume that we use the quantum Grover algorithm to solve the problem of "graph coloring" where every two adjacent nodes of a non-directed graph should get different colors. Additionally we look for graph coloring with the minimal number of colors. As we know, Grover algorithm will improve the complexity of finding exact minimum coloring from N to square-root-of N, where N is the number of all mappings of nodes to colors. Being able to solve this particular problem efficiently, most of the important optimization and decision problems that

appear in CAD algorithms could be solved. This universality of optimization algorithms and/or data structures is the general promise of:

(1) SAT solvers,

(2) universal algorithms,

(3) resolution-based programming languages such as Prolog, and

(4) hardware accelerators.

Some of these approaches, like the universal SAT solvers, are extensively used in modern CAD industry. And in our case, the universal solvers will be used in the work towards the promise of future "quantum CAD accelerators" that this thesis attempts to make a ground for.

Concluding on the "universality versus special domain" issue, this thesis develops algorithms that are both general, have both classical and quantum variants and allow in addition to incorporate problem specific knowledge into them.

1.6.4. The role of additional knowledge and heuristics in creating algorithms

It is well known that in classical algorithms, any additional knowledge about the problem, like for instance the upper bound to the chromatic number of the graph being colored, can help to create a more efficient algorithm. We will show that the same is true for quantum algorithms of certain type, including the quantum algorithm for

graph coloring. Among the several possible approaches to create such a meta-algorithm, the biologically motivated computations, such as evolutionary, were viewed as attractive because of their generality and flexibility. Thus, in the long research and development process to create QSPS and ECPS algorithms, the PSU group applied the biologically inspired, evolutionary processes of Genetic Algorithms and Genetic Programming. These algorithms were also sometimes combined with the, humanly designed, heuristic and search methods. This was done for instance in several papers by Karen Dill [Dill01], Martin Lukac [Lukac05a], and Normen Giesecke [Giesecke08].

The final quantum search algorithm presented here is motivated, however, more by the quantum mechanics than by the biology. On the other hand, our approach also points out to the ubiquity of some basic ideas in all of the Nature: Quantum Evolution versus Darwinian Evolution. Because of its general applicability and combination of problem-solving methods, the algorithm is denoted as the Quantum Search Problem Solver (QSPS). Its non-quantum counterpart is called Extended Cybernetic Problem Solver to underline the power of Cybernetics to unify the understanding and use of various mechanisms of Nature.

1.7. New integrated approaches to search

1.7.1. QSPS or Quantum Search Problem Solver

Within this dissertation, original automated quantum problem-solving methods are developed as its central point. Exploring evolutionary design and optimization techniques, investigating and discussing several design approaches as a potential candidate I decided how to combine them for the design of a meta-algorithm, as a quantum-mechanically and biologically inspired, application-specific reconfigurable parallel (FPGA-like) hardware. General search heuristics are utilized independently and in combination with other techniques. After several research approaches were investigated and analyzed together with my thesis advisor, a new type of meta-algorithm, with artificial evolutionary methodologies for algorithm development, combined with Heuristic Search, Constraint Programming, and human-designed Expert Systems, is created in this dissertation. This approach is referred to as the Quantum Search Problem-Solving Algorithm (QSPS). My approach supports quasi-automatic, and in future - automatic, design of application-specific quantum algorithms. These algorithms will be for logic synthesis, minimization, decision and other problems in quantum circuits, data mining, and other areas. The proposed approach is demonstrated on examples of binary logic, hardware circuit synthesis, "logic expression building" or Knowledge Discovery (i.e. explaining underlying principles by discovering meaningful patterns and rules about a data set), and logic minimization. Although we concentrate ourselves mainly with the new, proposed

here, area of Quantum Computer-Aided Design (chapters 11, 12, 13,14 and 15), this research can also be directly applied with broad implications to the fields of Intelligent Robotics (chapter 15), Machine Learning, Data Mining, and Evolvable Hardware. The wide applicability of our approaches results from the multiplicity of problems that can be characterized as the Constraint Satisfaction Problems.

1.7.2. Origins of our main quantum search idea

My first approach to develop a quantum search algorithm for CAD application originates from the paper by Lin, Thornton and Perkowski [Li06]. Their paper can be explained as an application of an exhaustive search speed-up by classical Grover algorithm to create the best Fixed Polarity Reed-Muller Form (FPRM). This circuit is found for a given truth table of positive and negative examples. These "examples" (terminology of machine learning) are called "minterms" in the area of logic synthesis. Such FPRM form is a type of structured logic expression that should be as simple as possible and should separate the truth from the false - thus for all minterm examples categorized as "false" (negative examples, zero-minterms) the value of expression is false and for all minterm examples categorized as "true" (positive examples or "ones" in the truth table) the value of the expression is true.

Careful analysis of the approach from [Li06], however, reveals that this idea can be applied with little modification to an incompletely specified function, thus

becoming applicable in Data Mining and Machine Learning [Dill01, Koza94, Koza99]. This observation was an inspiration and the starting point to much of the research reported in this thesis.

Let us explain the idea on a simple example of inducing formula from a set of examples.

A set of positive and negative examples is collected by observing successes or failures of various pairs of humans, related to their character, social position, physical properties, etc. Next an ideal life partner is induced from this set – it may be described by an expression "(Beautiful and Smart) ⊕ Rich". This formula means that the candidate person has to be either "beautiful and smart" or rich but not all positive properties at once: beautiful smart and rich (somebody who is beautiful, smart and rich may drop his partner soon, which was reflected in the particular set of specific examples given to the learning tool). Denoting B = Beautiful, S = Smart, and R = Rich, the learned (Fixed Polarity) Reed-Muller expression is BS ⊕ R. This is a Positive Polarity formula (i.e. all variables are not negated), possibly realized in a single (quantum) Toffoli gate. Thus, a logic formula that generalizes the results from all examples is the result of learning. This learning can be either classical Machine Learning or Quantum Machine Learning, presented in chapters 11 and 15.

My conclusion of this generalization is very powerful – *every problem for which*

"pure" Genetic Algorithm was applied to in binary logic synthesis or Constraint Satisfaction Problem can be extended now to a quantum search algorithm based on the Grover Algorithm or based on various generalization and variants of this algorithm that appeared subsequently and continue to appear in literature.

My approach applies also to the data that are incomplete (i.e. there are unknowns, or examples without positive or negative characterizations), thus we can construct or "build" a logical expression (in this case, the FPRM expression) to satisfy the behavioral criteria. As a new method of logic synthesis, the Quantum Search, as this one, offers a unique approach to automatic logic design or "quantum evolvable hardware". We can speculate that future evolvable hardware will be a quantum accelerator equipped with different "universal" components. These components will be in theory "universal" (such as SAT is universal in classical CAD) but practically only "wide range" components. These components will be created for particular applications such as: Grover Algorithm for FPRM, GRM, and ESOP synthesis, Quantum Walsh and Quantum Fourier Algorithms for image matching or spectral coefficients minimization for structured forms of learning, as well as other learning/problem-solving methods.

The "logic circuit" (equivalently, the "solution specification") is designed by evolutionary means and the process is entirely "hands-off" for the user. This is

however not a biological "Darwinian evolution", but the evolution of the superposed (quantum) states in a quantum computer. This computer is intentionally designed by humans to use quantum evolution in order to solve certain class of problems. Like in FPGAs, each class of problems requires new computer hardware, in this case a new oracle. The beauty of the proposed here method is that problems can be solved without explicit computer programming. It is just the use of the general quantum search algorithm itself. While the general search mechanism (of Grover) is universal, each specific problem is described by the user, as a specific oracle.

Observe that in future many parameterized descriptions of many problems will be created by designers, similarly as it is done now in the areas of "intellectual property design" and "circuit generators and hardware compilers" where sophisticated blocks are designed in hardware languages such as Verilog and VHDL. Because the oracles for classes of problems are similar, in the further future certain "smart software generators" will be written to create parameterized descriptions, similarly as it is done now in Matlab or AI-based generators of VHDL or Verilog programs. There exists therefore a clear path from the FPGA fast prototyping methods that are at the forefront of CAD tools in year 2008 to the future quantum CAD tools for quantum computers.
Further, let us observe that in theory, a single technique is applicable to solve all logic design problems (because all problems such as graph coloring or SOP minimization can be polynomially reduced to a Boolean Satisfiability formula - the SAT problem).

Such approach is theoretically feasible in classical computers, but only sometimes it is practical. Our quantum search has perhaps similar properties: although in theory we can reduce all problems to Quantum SAT, it is better when the user/designer disposes several types of reduction specified as software modules to be used.

In my thesis I explain in full all necessary details of quantum algorithms by Deutsch, Deutsch-Jozsa, Grover and other algorithms. This is done both for completeness of presentation in the thesis, but it is expected that the reader interested in all mathematical formalisms should study for instance the Chuang/Nielsen textbook [Nielsen00]. Starting from chapter 2, all our descriptions, however, have an ambition of being very simple, precise, and illustrative for the non-expert reader.

1.8. Summary of new concepts and ideas

Concluding, there are several new ideas proposed in my dissertation:

1. A new logic family of algebra is introduced. New families based on affine gates are used and new algorithms are created for them. The practicality of this new extension to Zhegalkin Logic is demonstrated. These algebraic forms realized as quantum arrays are highly testable.

2. These algebraic forms realized as quantum arrays are practically realizable in NMR and new quantum hardware technologies (as shown in Chapter 3).

3. The combined search strategies originally developed by Marek Perkowski [Perkowski82] and extended by James Brown [Brown90], Juling Lee [Lee99] and Karen Dill have been much extended, modified and implemented as the ECPS program and its behavior has been improved. It works regardless of the type of logic operators. Most importantly, this concept has been extended to parallel quantum searches.

4. Search algorithms for FPRM, KRM, GRM and ESOP circuits and general reversible circuits have been proposed. The algorithms are unaffected by the degree to which the problems were completely specified (i.e. a large or small number of "don't cares" is unimportant). This new learning meta-algorithm is the historically second quantum algorithm for logic synthesis, and the first algorithm for incompletely specified functions thus leading to Machine Learning applications. The goal is to demonstrate that the method is applicable to many benchmarks, for the logic minimization/synthesis of binary logic

hardware circuits, Knowledge Discovery for Data Mining, off-line Evolvable Hardware development, and Machine Learning.

5. The QSPS quantum evolvable hardware system has been invented and explained, it was also simulated using Matlab and a general-purpose quantum simulator QUIDPRO to prove the validity of our approach. We simulated the hardware for graph coloring, SAT, FPRM, GRM, ESOP and other circuit types, but for all algebraic extensions discussed in this thesis the same can be done.

1.9. Guide to the contents of chapters

This thesis takes background and ideas from several fields of physics, mathematics, computer science and computer engineering. We wanted also the dissertation to be as much as possible self-contained. My goal was predominantly to explain in an easy engineering text several complicated concepts that appeared so far only in mathematics, physics, or computer science journals and books. Unifying, simplifying and binding together were thus few of the tasks of this dissertation. Because several sub-areas of this thesis are interlinked to other sub-areas in many ways, organizing the text in a linear manner was not easy. Below we provide short information about what is in which chapter and how the chapters are interconnected.

1. Chapter 1 is an introduction to the presented research. It presents the history of this thesis and its main concepts from the bird's eye point of view – no details just basic concepts. However, to understand fully the contents of this chapter the reader should be familiar with the next chapters of the thesis first. Chapter 1 should be thus read again after the entire thesis.

2. Chapter 2 presents the design of quantum computers on the lowest level – electromagnetic pulses for NMR. Optimization of such circuits using search methods is presented and various basic gates are designed. This level of detail helps to formulate realistic costs of gates to be used in the next chapters. For instance, we learn how inexpensive are the quantum NOT and CNOT gates as compared to the quantum Toffoli gates. To demonstrate that our reversible logic synthesis methods from chapters 3 – 10 are general and applicable not only to NMR technology, we discuss how the Fredkin gate can be built using cellular automata.

3. In Chapter 3 we discuss the problems of designing larger quantum gates from small primitives and the links between low level and medium level synthesis of quantum (permutative) circuits. Chapter 3 is the background material for the thesis and it is based mostly on the literature. The concepts of basic expansions

in AND/EXOR logic, data structures and logic structures that are fundaments for the research of next chapters are introduced. Because the introduced in this thesis new structures such as the affine gates are the extensions of the existing structures, we present the basic material in all necessary detail.

4. In Chapter 4 devision trees and diagrams based on Davio expansions are introduced. The concept of polarity search is also explained in a simple way that will allow in chapters 7, 8, 9 to explain its use not only for FPRMs but for arbitrary linearly-independent families of forms.

5. Chapter 5 is an attempt to explain in as easy way as possible the concept of Grover algorithm, the central topic of this thesis. We explain first simple algorithms; Deutsch, Deutsch-Jozsa, Bernstein-Vazirani and Simon, making this way our explanation divided to several small pieces, each of them simpler to grasp. Finally the Grover algorithm is explained in full detail and from various points of view. This presentation is based on literature and previous work of PSU group. Certain intuitions which I found myself very useful to create new applications are also explained.

6. Chapter 6 introduces the concept of universal search method based on combining various search methods. This method is applicable to all

combinatorial synthesis problems and other problems introduced in this thesis. It can be used for both classical and quantum computing.

7. Chapter 7 is the main chapter of this thesis. It introduces the new powerful concept of affine gates and generator of circuits with such gates. All material in this chapter is new and is based on previous papers and reports of this author. The exhaustive and iterative-deepening depth-first search algorithms were implemented. Their complete analysis is given and superiority demonstrated. The invention of new quantum gates is documented.

8. Chapter 8 illustrates the use of universal search to the problem of synthesizing GRM forms for incompletely specified problems. This is the first solution to this problem. Analysis of experimental results is presented. Other applications of universal search are also explained.

9. Chapter 9 introduces GRMs, ESOPs and Linearly Independent logic. These expressions are used in the algorithms of this chapter. Multi-polarity Linearly independent expansions and Zhegalkin Logic are also presented, as much as necessary for the practical applications in quantum circuit design. Chapter 9 completes the first part of the thesis – designing various types of binary permutative quantum circuits (and also reversible circuits as a byproduct) using

various approaches based on exhaustive and intelligent searches. All these methods are useful for designing oracles which is the subject of next chapters. The methods are geared towards quantum oracles to give speedup to Grover algorithm, but in theory they can be used to build also classical oracles for reversible technologies, for instance in nano technology.

10. Chapter 10 introduces new multiple-valued gates and circuit structures and it generalizes some results of previous chapters to multiple-valued logic.

11. Chapter 11 is the link between the two parts of the thesis – first part about designing circuits and second part about designing algorithms (oracles) using blocks realized from these circuits. The chapter presents several practical blocks that are used to build oracles. Most of these blocks are next used in chapters 12 – 15 to construct oracles. All blocks are reversible and realizable with any quantum technology, but are optimized towards NMR-like technologies. The methods that we used to design these blocks are wider than the methods from the first part of the dissertation. The design of optimal oracles is a broad subject of study (it is also at its very beginning). The practical examples show that various circuit structures and design approaches are necessary to find the circuits that are reasonably small from the common-sense point of view. Design of quantum blocks, as in classical computing

requires human intuition and experience and can not be fully automated in 2008.

12. In Chapter 12 the concept of oracle design for Grover is illustrated with examples from Satisfiability, logic equations and logic design. A sub-family of reductions to SAT is illustrated, all of these problems are in close link to CAD algorithms from the thesis and many of them appear in synthesis algorithms from the first part of my dissertation.

13. Chapter 13 discusses Constraint Satisfaction problems as Grover Oracles. Specifically we illustrate the graph coloring problem.

14. Chapter 14 discusses crypto-arithmetic puzzles which are models of a wide category of problems in CAD, scheduling, planning, vision and robotics. Although we did not create a methodology to solve all problems of the classes from chapters 12 – 13, we collected enough ideas and examples to create a human-aided methodology of building algorithms and oracles that can be applied to solve new problems of these types.

15. Chapter 15 introduces first the class of constraint satisfaction problems in robotics and argues that the existence of a general tool to solve all these

problems more efficiently would mean a breakthrough in robotics. Next we introduce the adiabatic quantum computer Orion, recently presented by a commercial company DWAVE for the first time in history and we present that our model can be reduced to their model, making all approaches from this thesis practically applicable to an existing quantum (adiabatic) computer which can be used in robotics applications. This part of the chapter is a compilation of many ideas from Internet and recent papers from years 2005 - 2008. Finally we present a new class of problems in Machine Learning and Data Mining which use multi-polarity spectral transforms with Grover algorithm selecting the best polarity (a well-known exponential complexity problem). These models can be also reduced to the adiabatic computer. Some speculations on future research based on these ideas are also given.

16. Finally Chapter 16 is a comprehensive conclusion of my thesis. We present new ideas that expand the methods and ideas from this thesis and we speculate on future research in this area.

The Visual Flow Diagram of the chapters in the thesis is given in Figure 1.1.

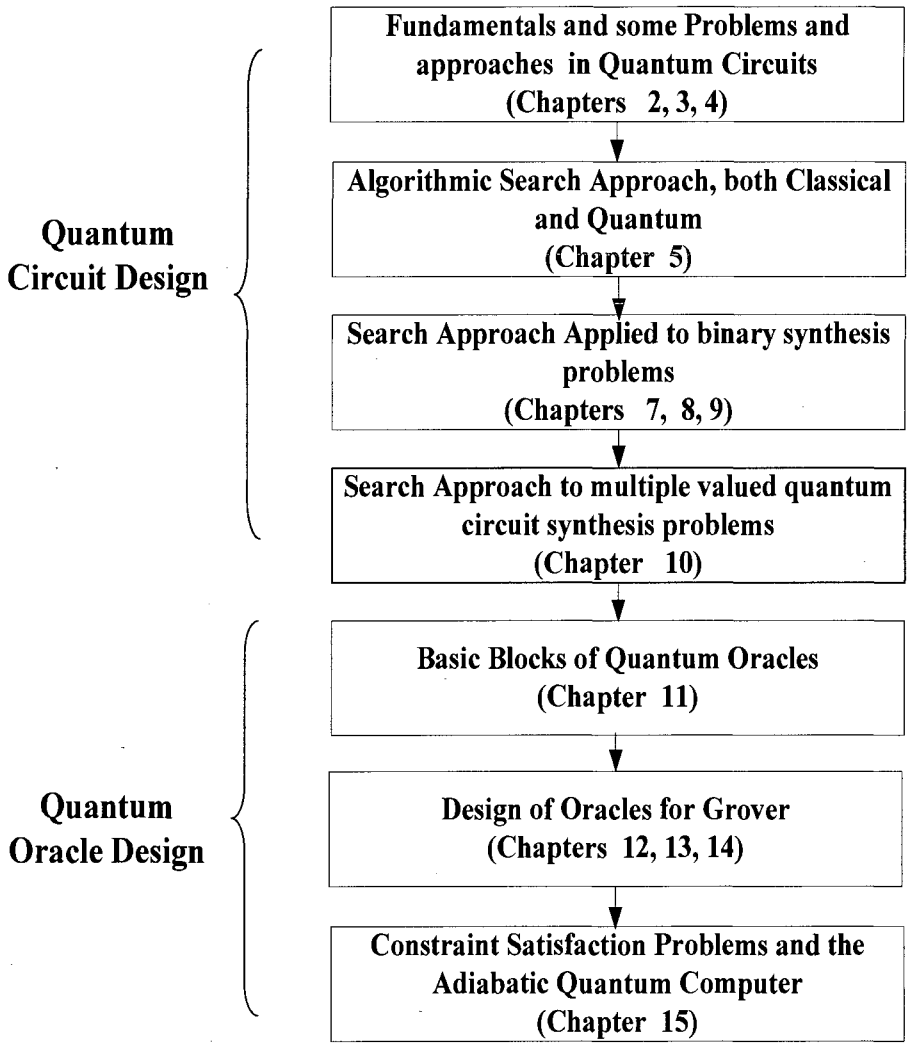


Figure 1.1: The contents of the Chapters.

The comparison of methods and algorithm approaches from the thesis is given in Table 1.1.

Algorithms / Problems	Linear Search	Tree Search	Evolutionary Search	ECPS Hybrid Search	QSPS Quantum Search
FPRM	Chapters 3 and 4	Chapters 3 and 4	Not known	Chapter 9	Chapter 15
ESOP	Does not exist	Chapter 6	Not known	Chapters 6 and 12	Chapter 12
Affine Generator	Old version: Exhaustive search (chapter 7)	Chapter 7	Does not exist	Future work	Future work
GRM	Zheng [Zeng95]	Sasao,Debnath,Dill [Sasao95, Debnath98, Dill01]	Dill [Dill98]	Chapter 8	Chapter 12
Graph coloring	Does not exist	Popular		Chapter 6	Chapter 13
SAT	Does not exist	Popular		Chapters 6 and 12	Chapter 12
Set Covering	Does not exist	Popular		Chapters 6 and 12	Chapter 12

Table 1.1: Various approaches to main synthesis problems of the thesis.

CHAPTER 2

From Realization Technology Models of Quantum Permutative Gates to Uniform Synthesis Approaches.

2.1. Towards Computer Aided Design of Quantum Computers.

In the past few decades, integrated circuit technology has grown substantially, from that of realizing only a few logic gates on a single device to more currently constructing billions of logic gates, effectively creating a "computer on a chip". But, with this massive growth in technological capability, it becomes ever more difficult for future human invention to both maintain and surpass the capabilities of previous creations. With larger and larger designs, a need for high testability, increasing design complexity, and more aggressive time-to-market schedules, higher demands are placed on the human design team. Thus, new methods of invention become necessary, which combine both human expertises with that of the increased computational capabilities of computers. As already discussed in chapter 1, one of the most promising technologies of the future is the quantum computer. Computer Aided Design methods should be developed for quantum computers as well as methods of efficient mapping of quantum algorithms to quantum hardware. Our synthesis will be mostly on a "permutative" level of gates such as Toffoli and Fredkin. Thus in this chapter we will present design details close to technologies. In sections 2.2 and 2.3 we will illustrate two respective ways of building such gates on top of the two types of lower level primitives.

2.2. Quantum gates and circuits on the level of pulses in Quantum technologies such as NMR and ion traps.

2.2.1. The quantum gates on the level of electromagnetic pulses. The fundaments.

Every quantum gate and circuit, from the smallest like Pauli rotation to the largest, like the Quantum Algorithm of Grover, can be represented by a unitary matrix of the Hilbert space. Hilbert space can be defined as vector space with infinitely many dimensions. The dot product u · v = Sum u_i v_i (or Sum u_i v_i* for spaces on the complex numbers) is replaced by the corresponding infinite sum or integral. The (length of u)^2 is defined as **u** · **u** as usual, and the space only contains those vectors whose length is finite. We are building large quantum matrices of algorithms from small quantum matrices of gates (pulses) that are realizable in some selected quantum technologies. In this chapter we will concentrate on realization of quantum circuits in two most advanced as of 2007 quantum realization technologies: that of liquid state nuclear magnetic resonance (NMR) [Cory97, Gershenfeld97, Jones98, Jones98a] and ion traps [Leibfried03, Paul90, Steane97, Wineland98].

As we will see in next chapters, the designer designs a quantum algorithm on many levels; block level, circuit level and gate level. This requires the decomposing or composing of the gates' Unitary matrices. Here we will start with standard elementary

quantum gates for computation [Lee06]. In implementation, each gate is again converted to a sequence of physical operations that a given type of quantum computer technology can actually directly implement. In this thesis we abstract from the physical nature of qubits, they may be spins of electrons or polarization of photons. This is completely immaterial to the quantum mechanics abstraction used in this dissertation.

The total calculation time in quantum computation depends on the number of basic gates in the series and the number of physical operations required for a quantum system to implement each gate. Let us denote a series of physical operations as a sequence of *electromagnetic pulses* distinguishing it from the series of basic gates, as the physical operations are either the time evolution of finite duration under the influence of an externally applied magnetic field, or interactions between qubits. In quantum computation, the calculation time is a very precious resource due to the finite coherence time of a quantum system. Therefore, it is important to know the cost of gates for the successful implementation of an algorithm, and thus for the future design of a practical quantum computer. Once the pulse sequences for the single-qubit and two-qubit gates are obtained, the total pulse sequence for a circuit is given by replacing each elementary gate by the corresponding pulse sequence. The pulse sequence of more complicated circuits with larger numbers of input qubits can be obtained in the same way, that is, by finding the quantum circuits composed of simpler gates and replacing each gate by the corresponding pulse sequence. In paper [Lee06]

the costs of gates were calculated in terms of numbers of basic pulses. The software used there calculated the cost of each gate by reducing the number of pulses in the sequence using the commutation rules of the pulse operations using naïve greedy search algorithm. We demonstrate that these results can be improved by using the new heuristic search algorithm that will be developed in this chapter.

The optimized circuits presented in [Lee06] are not necessarily minimal, since the heuristic algorithm that found them has no way of knowing if the solutions found are local or global minima. Therefore, they may not be the true minimal costs of gates, and the authors claim only to provide the upper bounds as the worst case. To evaluate the quality of their heuristic algorithm we develop exhaustive search to be used in comparison of small problems.

We know that Quantum Computation relies on quantum mechanics which is a mathematical model that describes the evolution of physical realization of computation and hence the computer itself. Several philosophically different but physically equivalent formulations have been found for quantum mechanics [Styer02]. In this thesis, we follow Schrödinger [Schrödinger26] which describes the physical state of a quantum system by a temporally evolving vector $|\psi(t)\rangle$ in a complete complex inner product space H called a Hilbert space. The time evolution under the influence of a single term of the Hamiltonian is a single physical operation, in this dissertation we will be optimizing circuits at the level of such operations (pulses).

(Hamiltonian is a physical state of a system which is observable corresponding to the total energy of the system. Hence it is bounded for finite dimensional spaces and in the case of infinite dimensional spaces, it is always unbounded and not defined everywhere.). The new approach in quantum circuits synthesis introduced in [Lee06] differed from the previous publications [Smolin96, Shende03, Miller02, Miller03] which optimized the quantum circuit at higher levels of abstraction. It is still rare to see papers in the literature that would optimize on the level of pulses, but this is done systematically in this thesis. This is partially possible thanks to our software which is intended to perform hierarchical top-down synthesis from various levels of specification. In one synthesis variant, the software will modify the initial non-optimal design by shifting gates left and right in the circuit and applying quantum logic identities, analogously to [Lee06, Lomont03], but calculating the combined cost of the operations that are necessary to build arbitrary quantum circuits instead of the total gate cost (gate number). The approach from [Lee06] was next extended to larger circuits, but with a smaller number of transformations [Miller03], the so-called "template matching approach". In next chapters we present software that operates on larger circuits and with a larger, user-defined numbers of operations.

The most important result from [Lee06] is a table of realizations of useful gates and their costs, given in Table 2.2.1

Table 2.2.1 - Cost of gate primitives

Gate	Pulse Sequence	Cost
NOT	$iR_x(\pi)$	1
Phase	$e^{i\frac{\phi}{2}}R_z(\phi)$	1
Hadamard	$iR_y(\frac{\pi}{2})R_z(\pi)$	2
CNOT	$e^{-i\frac{\pi}{4}}R_{2y}(\frac{\pi}{2})R_{1z}(-\frac{\pi}{2})R_{2z}(-\frac{\pi}{2})J_{12}(\frac{\pi}{2})R_{2y}(-\frac{\pi}{2})$	5
SWAP	$e^{-i\frac{3\pi}{4}}R_{2y}(\frac{\pi}{2})R_{2z}(-\frac{3\pi}{2})R_{1z}(-\frac{3\pi}{2})J_{12}(\frac{\pi}{2})R_{2y}(\frac{\pi}{2})$ $R_{1y}(-\frac{\pi}{2})J_{12}(\frac{\pi}{2})R_{1x}(\frac{\pi}{2})R_{2x}(-\frac{\pi}{2})J_{12}(\frac{\pi}{2})$ $R_{2y}(-\frac{\pi}{2})$	11
Peres	$e^{-i\frac{\pi}{8}}R_{3y}(\frac{\pi}{2})R_{3z}(\frac{\pi}{4})R_{2z}(-\frac{3\pi}{4})R_{1z}(-\frac{\pi}{4})J_{23}(\frac{\pi}{4})$ $R_{2x}(-\frac{\pi}{2})J_{12}(\frac{\pi}{2})R_{2y}(-\frac{\pi}{2})R_{2z}(\frac{\pi}{4})J_{31}(-\frac{\pi}{4})$ $J_{23}(-\frac{\pi}{4})R_{3y}(-\frac{\pi}{2})$	12
Toffoli	$e^{-i\frac{\pi}{8}}R_{1z}(-\frac{\pi}{4})R_{2z}(-\frac{\pi}{4})J_{12}(\frac{\pi}{4})R_{3y}(-\frac{\pi}{2})$ $R_{3z}(\frac{\pi}{4})J_{31}(-\frac{\pi}{4})J_{23}(-\frac{\pi}{4})R_{3y}(\frac{\pi}{2})J_{31}(\frac{\pi}{2})$ $R_{3x}(\frac{\pi}{2})J_{23}(-\frac{\pi}{4})R_{3x}(-\frac{\pi}{2})J_{31}(-\frac{\pi}{2})$	13

Fredkin	$e^{-i\frac{7\pi}{8}} R_{2y}(\frac{\pi}{2}) R_{1z}(-\frac{3\pi}{4}) J_{23}(\frac{\pi}{2}) R_{3x}(\frac{\pi}{2}) R_{3z}(-\frac{3\pi}{4})$ $J_{12}(\frac{\pi}{2}) R_{2y}(\frac{\pi}{2}) R_{2z}(-\frac{7\pi}{4}) J_{23}(\frac{\pi}{4}) R_{2x}(-\frac{\pi}{2})$ $J_{12}(\frac{\pi}{2}) R_{2y}(-\frac{\pi}{2}) R_{2z}(-\frac{\pi}{4}) J_{31}(-\frac{\pi}{4}) J_{23}(-\frac{\pi}{4})$ $R_{3x}(\frac{\pi}{2}) R_{2x}(-\frac{\pi}{2}) J_{23}(\frac{\pi}{2}) R_{2y}(-\frac{\pi}{2})$	19
Miller	$e^{i\frac{7\pi}{8}} R_{2y}(\frac{\pi}{2}) J_{12}(\frac{\pi}{2}) R_{2x}(-\frac{\pi}{2}) R_{2z}(-\frac{5\pi}{4}) R_{1x}(\frac{\pi}{2})$ $J_{31}(\frac{\pi}{2}) R_{1y}(\frac{\pi}{2}) R_{1z}(-\frac{7\pi}{4}) R_{3x}(\frac{\pi}{2}) R_{3z}(-\frac{5\pi}{4})$ $J_{23}(\frac{\pi}{4}) R_{2x}(-\frac{\pi}{2}) J_{12}(\frac{\pi}{2}) R_{2y}(-\frac{\pi}{2}) R_{2z}(-\frac{\pi}{4})$ $J_{31}(-\frac{\pi}{4}) J_{23}(-\frac{\pi}{4}) R_{3y}(\frac{\pi}{2}) R_{2x}(-\frac{\pi}{2}) R_{1x}(-\frac{\pi}{2})$ $J_{23}(\frac{\pi}{2}) R_{2y}(-\frac{\pi}{2}) J_{31}(\frac{\pi}{2}) R_{1y}(-\frac{\pi}{2})$	24

The basic quantum gates that are used in quantum circuits are Inverter (NOT, Pauli X rotation), Hadamard, Toffoli, Feynman, CV (controlled square root of NOT) and $CV^{\dagger}$ (controlled square root of NOT Adjoint gate). *These gates are truly quantum and universal.* Their subset {NOT, CV, $CV^{\dagger}$} allows creating all permutative binary quantum gates (circuits) by their compositions.

A quantum gate operating in parallel with another quantum gate will increase the dimensions of the quantum logic system represented in matrix form. This is due to

application of the Kronecker (tensor) product of matrices to the system. Kronecker Matrix Multiplication is responsible for the growth of qubit states such that N bits correspond to a superposition of r^N states, whereas in other digital systems, N bits correspond to r^N distinct states. The number r denotes the base (radix) of logic, being 2 for the binary and 3 for the ternary logic. The Kronecker Product of two one-qubit gates is illustrated below:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} x & y \\ z & v \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} x & y \\ z & v \end{bmatrix} & b\begin{bmatrix} x & y \\ z & v \end{bmatrix} \\ c\begin{bmatrix} x & y \\ z & v \end{bmatrix} & d\begin{bmatrix} x & y \\ z & v \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ax & ay & bx & by \\ az & av & bz & bv \\ cx & cy & dx & dy \\ cz & cv & dz & dv \end{bmatrix}$$

(Equation 2.2.1)

The multiplication operation in Equation 2.2.1 is, in the most general case, the multiplication of complex numbers. Kronecker multiplication can be defined for matrices or vectors of any sizes.

A quantum gate in series with another quantum gate will retain the dimensions of the quantum logic system. The resultant matrix is calculated by multiplying the operator matrices in a reverse order. This is standard multiplication operation on matrices.

In Figure 2.2.1 we show the notation and the unitary matrix of a very important quantum gate – the Hadamard gate. This is a "truly quantum" gate that cannot be realized in a "binary reversible" or "permutative" circuit. This is in contrast to the

permutative gates (described by permutative matrices) that can be realized by standard reversible logic circuits. This realization is however only in logic/mathematic sense and the reversible gates do not allow superposition and entanglement.

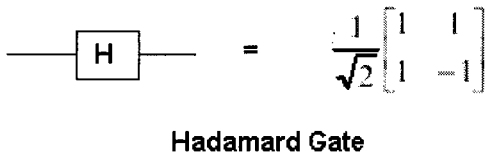


Figure 2.2.1: Hadamard gate notation and its unitary matrix.

An example of a unitary and permutative matrix is the Feynman gate. A permutative matrix has exactly one '1' in every row and column.

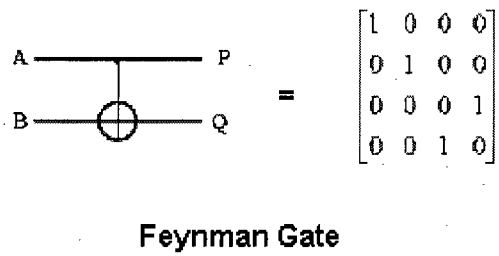


Figure 2.2.2: Feynman gate notation and its unitary (in this case also permutative) matrix

By V we denote the "square root of NOT" gate. When it is applied to basis states then it creates superposition states on its output. The conjugated transpose of a unitary matrix U is called the adjoint of matrix U and denoted by $U^{\dagger}$. By $V^{\dagger}$ we denote a gate that has a unitary matrix which is an adjoint of V. Therefore, the adjoint of V is called "square root of NOT adjoint" and has the unitary matrix $U_{V^{\dagger}}$ of gate $V^{\dagger}$. Design of

many permutative gates is based on (controlled) cascading of V and $V^{\dagger}$ gates. Cascading two square-root-of-NOT gates acts as a basic inverter gate (see Figure 2.2.3a). The operation of the circuit from Figure 2.2.3a can be explained by the matrix multiplication. Multiplying the unitary matrix U_V by the input state we obtain the vector $\frac{1}{2}$ [1+j 1-j] ^{T} = V_0, Figure 2.2.9a. By multiplying V by this vector we obtain vector [0 1] ^{T} = $|1\rangle$. The kets $|0\rangle$ and $|1\rangle$ are in Dirac notation and they represent classical 0 and the classical 1 in quantum mechanics.

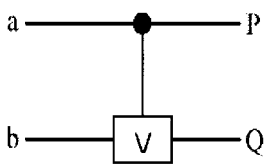
We have also that

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$$

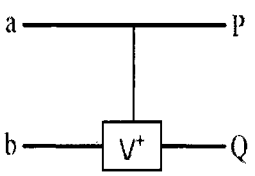
$$\textit{and} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T.$$

(a) 

$\Leftrightarrow \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

(b) 

$\Leftrightarrow \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+j}{2} & \frac{1-j}{2} \\ 0 & 0 & \frac{1-j}{2} & \frac{1+j}{2} \end{bmatrix}$

(c) 

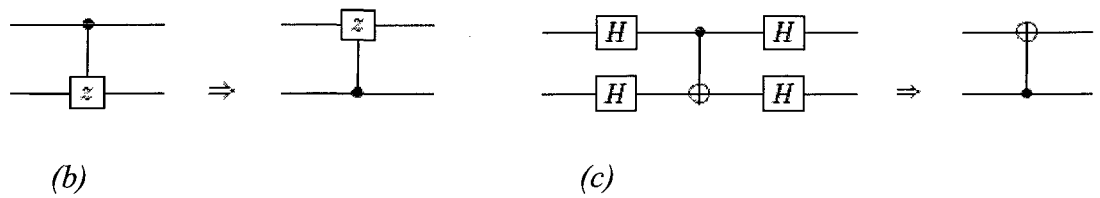
$\Leftrightarrow \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1-j}{2} & \frac{1+j}{2} \\ 0 & 0 & \frac{1+j}{2} & \frac{1-j}{2} \end{bmatrix}$

Figure: 2.2.3 (a) Cascading V gates creates an inverter. Measurement of intermediate state would give $|0\rangle$ and $|1\rangle$ with equal probabilities, composition of these gates acts as a classical inverter (b) Controlled-V gate and its unitary matrix,(c) Controlled-$V^{\dagger}$gate and its unitary matrix.

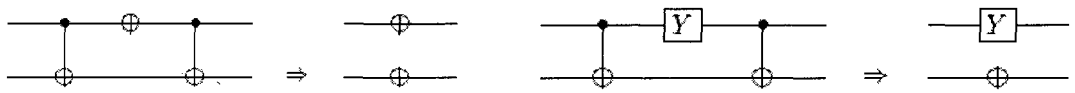
A quantum circuit can be easily analyzed. A parallel connection of gates corresponds to the Kronecker Product (the Tensor Product) of unitary matrices of respective gates. The serial connection of gates corresponds to the serial multiplication (in reverse order) of the matrices of these gates. One can check that the equivalence transformations from Figures 2.2.4 and 2.2.5 are correct. All verifications of quantum equivalence transformations can be done by multiplying and comparing respective unitary matrices. Figure 2.2.6a presents the controlled general phase gate used together with a pair of the pseudo-Hadamard and its inverse gate. Figure 2.2.6b has the symbolic unitary matrix when the control signal is $|1\rangle$. By substituting various values of angles, 0°, 90°, -90°, 180° the unitary matrices are created which are next combined with the pseudo-Hadamard matrices, as in Figure 2.2.6b. This leads to the table from Figure 2.2.6c that demonstrates that by changing the angle the gate from Figure 2.2.6a can work as a 2-qubit identity, controlled-V, controlled-$V^{\dagger}$ and CNOT. Actually this gate can be used as a controlled root of various degrees. Figure 2.2.6d illustrates unitary matrices for various angles of Y. This figure demonstrates therefore the usefulness of Y and Z rotations to create gates.

$$Y(\phi)=\cos\frac{\phi}{2}I-i\sin\frac{\phi}{2}Y=\cos\frac{\phi}{2}\begin{bmatrix}1&0\\0&1\end{bmatrix}-i\sin\frac{\phi}{2}\begin{bmatrix}0&-i\\i&0\end{bmatrix}=\begin{bmatrix}\cos\frac{\phi}{2}&0\\0&\cos\frac{0}{2}\end{bmatrix}-\begin{bmatrix}0&-i^2\sin\frac{\phi}{2}\\i^2\sin\frac{\phi}{2}&0\end{bmatrix}=\begin{bmatrix}\cos\frac{\phi}{2}&-\sin\frac{\phi}{2}\\\sin\frac{\phi}{2}&\cos\frac{\phi}{2}\end{bmatrix}$$

(a)

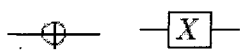


(b) (c)

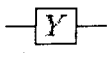


(d) (e)

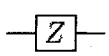
Figure 2.2.4: (a) Example how to calculate unitary matrices of generalized rotations from general matrix formulas in Table 2.2.1. (b) Equivalent transformation of Z gate, (c) equivalent transformation of CNOT and Hadamard gates, (d) CNOT and NOT transformation, (e) CNOTs and Pauli Y transformation.



$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

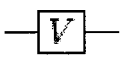


$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

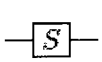


$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$



$$V = \frac{1}{2} \begin{pmatrix} 1+i & 1-i \\ 1-i & i+1 \end{pmatrix}$$



$$S(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\varphi} \end{pmatrix}$$

Figure 2.2.5a: Basic gates: NOT (or Pauli X), Pauli Y, Pauli Z, Hadamard, Controlled Square Root of NOT or V, Phase Gate.

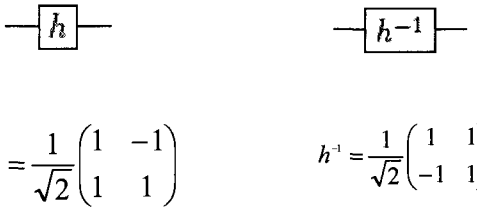


Figure 2.2.5b: Pseudo-Hadamard and inverse pseudo-Hadamard gates.

We will be using the single qubit gates that are commonly used in papers on quantum synthesis. They are : NOT (Pauli rotation X, denoted also in literature by σ_x), Hadamard, Phase, and T. Some of these gates are shown in Figure 2.2.5a. Some gates are also listed in Tables 2.2.1 and 2.2.2. We use Pauli rotations X, Y and Z or arbitrary angle rotations with respect to axes X, Y and Z of the Bloch sphere and some their special cases for fixed angles which are multiples of 45°. We will use also two new gates; pseudo-Hadamard h and its adjoint pseudo-Hadamard gate h^{-1}, Figure 2.2.5.b, because they are used to build many quantum gates, both permutative (pseudo-binary) and general-purpose-quantum gates (called also truly quantum gates) that are most useful in synthesis [Biamonte04, Jones98, Jones98a].

$$T=\begin{bmatrix}1 & 0\\ 0 & e^{\frac{j\pi}{4}}\end{bmatrix},\quad P(\phi)=e^{\frac{j\phi}{2}}I,\quad X(\phi)=\cos\frac{\phi}{2}I-j\sin\frac{\phi}{2}X,$$

$$Y(\phi)=\cos\frac{\phi}{2}I-j\sin\frac{\phi}{2}Y,\quad Z(\phi)=\cos\frac{\phi}{2}I-j\sin\frac{\phi}{2}Z$$

Table 2.2.2: X,Y,Z Pauli phase rotations.

In Table 2.2.2 symbols X, Y, and Z are the defined earlier Pauli spin matrices and P(ϕ), X(ϕ), Y(ϕ), and Z(ϕ) are the corresponding 2*2 matrices of arbitrary parameterized angle rotations by angle ϕ. The rotations X(ϕ), Y(ϕ), and Z(ϕ) can be explained as rotations with respect to angles X, Y and Z, respectively, as illustrated on the Bloch sphere [Nielsen00]. P is a phase rotation by ϕ/2 to help match identities automatically and its idea comes from [Lomont03].

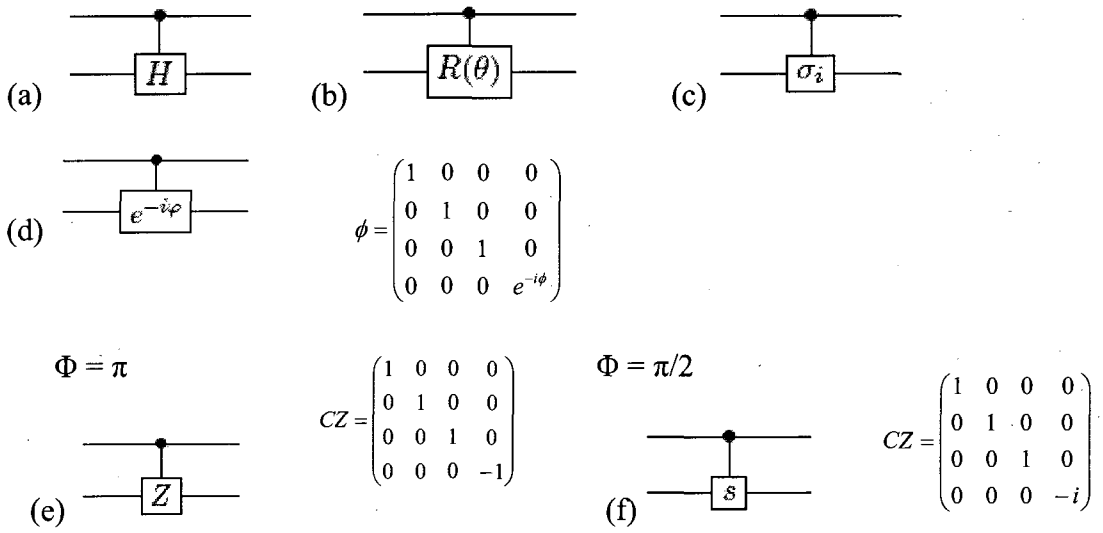
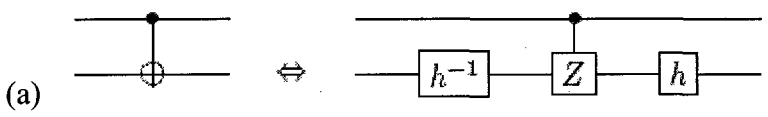


Figure 2.2.6: Controlled gates. (a) Controlled Hadamard gate, (b) Controlled Rotation with respect to angle θ. This symbol applies to any angle, particularly X, Y and Z. Additional symbol is used to denote the angle, (c) symbol of Pauli rotation where subscript i = X,Y,Z, (d) controlled phase and its unitary matrix, (e) Controlled Z and its unitary matrix, (f) controlled phase gate and its unitary matrix.



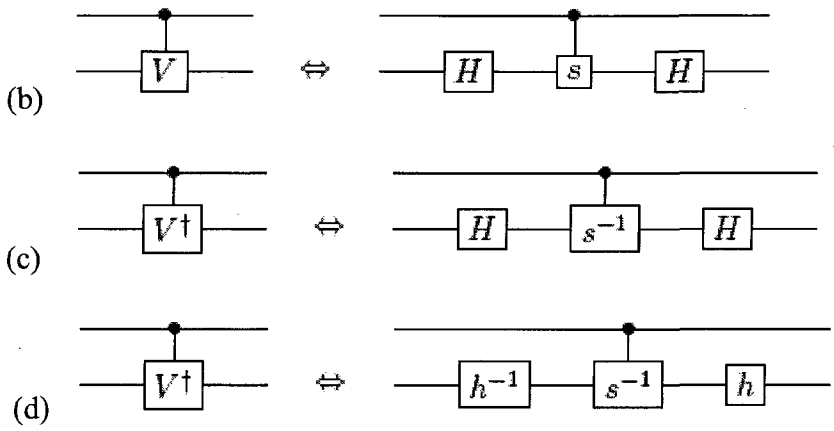
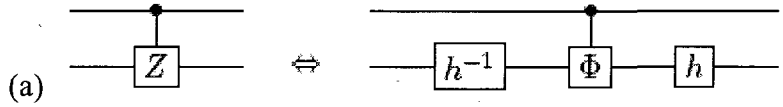


Figure 2.2.7: (a) CNOT realized with controlled-Z and pseudo-hadamard gates. Symbol h stands for pseudo-hadamard gate and symbol h^{-1} for inverse pseudo-hadamard gate. (b) CV realized with Controlled-S and Hadamard gates, (c) $CV^{\dagger}$ realized with controlled-S^{-1} and Hadamards, (d) $CV^{\dagger}$ realized with controlled-S^{-1} and pseudohadamards. Observe that this realization requires less pulses than its equivalent from Figure 2.2.7c.



(b)

$$h*\phi*h^{-1} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix} * \begin{bmatrix}1 & 0\\ 0 & e^{i\phi}\end{bmatrix} * \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ -1 & 1\end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix}1+e^{i\phi} & 1-e^{i\phi}\\ 1-e^{i\phi} & 1+e^{i\phi}\end{bmatrix}$$

(c)

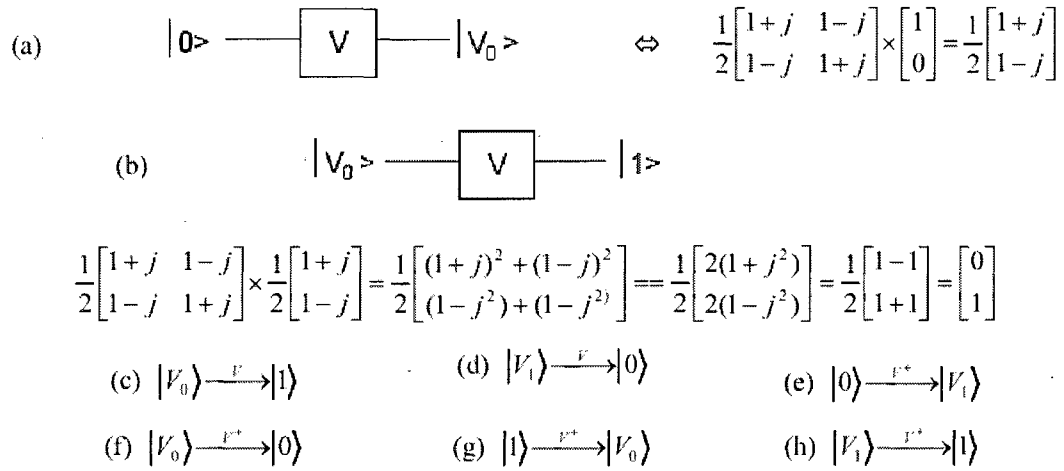
$\phi = 0$	I
$\phi = 90^0$	V
$\phi = -90^0$	V^+
$\phi = 180^0$	X

(d)

$\phi = 0$	I
$\phi = 90^0$	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = h$
$\phi = -90^0$	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = h^{-1}$
$\phi = 180^0$	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

Figure 2.2.8: (a) Controlled-Z gate realized with controlled-phi gate surrounded by pseudo-hadamards, (b) Calculation of unitary matrix for lower qubit of this gate, (c) Various gates realized by ϕ for angles 0^o, 90^o, -90^o and 180^o in X rotations. The ϕ gate realizes identity, Square-root-of-NOT, its adjoint and Inverter, (d) gates realized by Y rotations.

Let us now try to find, by matrix/vector multiplication, all possible states that can be created by applying all possible serial combinations of gates V and $V^\dagger$ to states $|0\rangle, |1\rangle$ and all states created from these basis states (Figure 2.2.9). A qubit $|0\rangle$, given to a "square root of NOT" gate (Figure 2.2.9a) gives a state denoted by $|V_0\rangle$. After measurement this state gives $|0\rangle$ and $|1\rangle$ with equal probabilities ½. Similarly all other possible cases are calculated in Figure 2.2.9b – h. As we see, after obtaining states $|0\rangle, |1\rangle, |V_0\rangle$ and $|V_1\rangle$ the system is closed and no more states are generated. Therefore the subset of (complex, continuous) quantum space of states is restricted with these gates to a set of states that can be described by a four-valued algebra with values $\{|0\rangle, |1\rangle, |V_0\rangle, |V_1\rangle\}$.



$$\frac{1}{2}\begin{bmatrix}1+j & 1-j\\ 1-j & 1+j\end{bmatrix}\times\frac{1}{2}\begin{bmatrix}1+j\\ 1-j\end{bmatrix}=\frac{1}{2}\begin{bmatrix}(1+j)^2+(1-j)^2\\ (1-j^2)+(1-j^2)\end{bmatrix}==\frac{1}{2}\begin{bmatrix}2(1+j^2)\\ 2(1-j^2)\end{bmatrix}=\frac{1}{2}\begin{bmatrix}1-1\\ 1+1\end{bmatrix}=\begin{bmatrix}0\\ 1\end{bmatrix}$$

(c) $|V_0\rangle \xrightarrow{V} |1\rangle$ (d) $|V_1\rangle \xrightarrow{V} |0\rangle$ (e) $|0\rangle \xrightarrow{V^+} |V_1\rangle$

(f) $|V_0\rangle \xrightarrow{V^+} |0\rangle$ (g) $|1\rangle \xrightarrow{V^+} |V_0\rangle$ (h) $|V_1\rangle \xrightarrow{V^+} |1\rangle$

Figure 2.2.9: Calculating all possible superposition states that can be obtained from basis states $|0\rangle$ *and* $|1\rangle$ *using V and* $V^\dagger$ *gates.*

Figure 2.2.3b shows the Controlled-V gate (Controlled-Square-Root-of-Not, denoted also by CV) and its unitary matrix. The gate operates as follows. Control signal *a* goes through the gate unaffected, i.e. $P = a$. If the control signal has value 0 then the qubit *b* goes through the controlled part unaffected, i.e. $Q = b$. If $a = 1$ then the unitary operation that is inside the box is applied to the input signal b, it means $Q = V$ (b) in our case. This operation is general for all binary controlled gates, for instance the Controlled-V-adjoint (Controlled-Square-root-of-Not-adjoint, denoted also by $CV^\dagger$). This gate and its unitary matrix are shown in Figure 2.2.3c.

2.2.2. Models of Basic Gates

The component particles of the quantum system should have at least two well-defined quantum states to be used as qubits, and should interact with each other. There must be a way for external devices to perform single-qubit operations and read the qubit

states. When we write "particles" here it may be any quantum entities: electrons, protons, other particles, photons, ions, nuclei atoms etc. We define a model quantum computer for both NMR and ion trap technologies as a system that meets the following specifications:

The Hamiltonian of the system is given by,

$$H = \sum_{i,\alpha} a_{i\alpha}\sigma_{i\alpha} + \sum_{i,j} J_{ij}\sigma_{iz}\sigma_{jz}$$

Equation 2.2.2.1

where α = x, y or z and symbol σ represent one of the Pauli operators. This is the most familiar form of the spin Hamiltonian where spins are interacting with each other in an external magnetic field. However, this Hamiltonian is not particular to spin systems but is general, as similar forms are relevant for any quantum computer. It is common to refer to the first term of the Hamiltonian in Equation 2.2.2.1 as the Zeeman term. The second term as of Equation 2.2.2.2 is known as the interaction term.

$$\sum_{i,\alpha} a_{i\alpha}\sigma_{i\alpha}$$

Equation 1.2.2.2 – Zeeman term

The Zeeman term is necessary to produce all the single qubit gates. As its name implies the second term defines interaction between qubits, such as those that are essential to make a CNOT gate. In the standard form, the so-called Ising model [Lee06] is characterized by the interaction of only Z-components of spins. The interaction form should not necessarily be of the Ising type, although it is advantageous because the Ising type interaction can generate the indispensable CNOT gate, while it is not quite clear if general interactions can do the same.

The physical pulses one is able to implement by the Hamiltonian from Equation 2.2.2.2 are,

$$R_{i\alpha}(\phi) = e^{-i\frac{\phi}{2}\sigma_{i\alpha}}$$

$$J_{ij}(\phi) = e^{-i\frac{\phi}{2}\sigma_{iz}\sigma_{jz}}$$

Equation 2.2.2.3

We define the cost of a gate as the number of pulses corresponding to the minimal pulse-level implementation of this gate. The algorithm first introduced by Soonchill Lee [Lee06] and also our improved algorithm perform reduction (full or partial) using the commutation rules on the sequence of pulses representing the gates circuit.

2.2.3. Circuit Identities and Optimizing Transformations

The reduction uses, among others the well-known rule [Nielsen97]: [A, B] = AB – BA, AB-BA = 0 → AB = BA.

This reduction rule is illustrated in the quantum circuit from Figure 2.2.3.1 which means, that one can shift left or right pulses or gates for which the above rule holds.

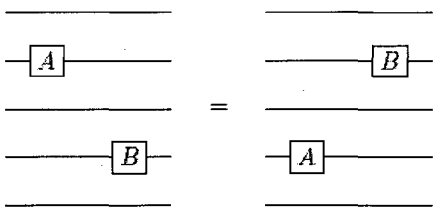


Figure 2.2.3.1: Graphical illustration of the rule [A, B] = 0.

In my reduction algorithm the following commutation rules are used (Equations 2.2.3.1 – 2.2.3.4):

$$[R_{i\alpha}, R_{j\alpha'}] = 0 \text{ for } i \neq j \qquad \textit{Equation 2.2.3.1}$$

$$R_{ix}(\pm\pi)R_{iy}(\phi) = R_{iy}(-\phi)R_{ix}(\pm\pi)$$

$$R_{ix}(\phi)R_{iy}(\pm\pi) = R_{iy}(\pm\pi)R_{ix}(-\phi)$$

$$R_{ix}(\pm\frac{\pi}{2})R_{iy}(\phi) = R_{iz}(\pm\phi)R_{ix}(\pm\frac{\pi}{2})$$

$$R_{ix}(\pm\frac{\pi}{2})R_{iz}(\phi) = R_{iy}(\mp\phi)R_{ix}(\pm\frac{\pi}{2})$$

$$R_{ix}(\phi)R_{iy}(\pm\frac{\pi}{2}) = R_{iy}(\pm\frac{\pi}{2})R_{iz}(\pm\phi)$$

$$R_{ix}(\phi)R_{iz}(\pm\frac{\pi}{2}) = R_{iz}(\pm\frac{\pi}{2})R_{iy}(\mp\phi) \qquad \textit{Equation 2.2.3.2}$$

and the relations generated by the cyclic permutation of x → y → z.

$$[J_{ij}, J_{i'j'}] = 0 \qquad \textit{Equation 2.2.3.3}$$

$$[J_{ij}, R_{i'z}] = 0 \qquad \textit{Equation 2.2.3.4}$$

Graphically, they are represented as in Figure 2.2.3.2. If necessary, more rules can be added to the program, and/or we can make some rules usable only in one direction (only from left to right or from right to left).

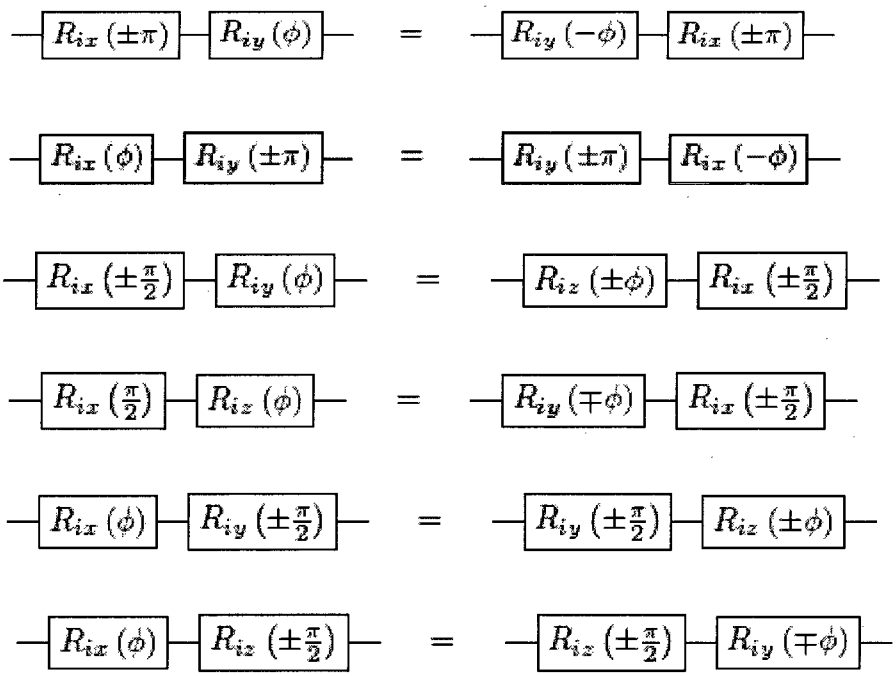


Figure 2.2.3.2: Graphical illustration of some commutation rules for quantum algebra that are used in my tree search-based pulse-level circuit minimization algorithm.

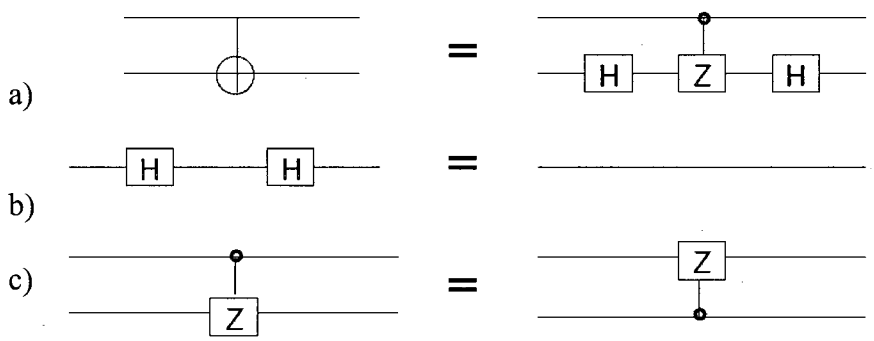


Figure 2.2.3.3 : (a) The Controlled-NOT gate realised by controlled-Z gate surrounded by Hadamard gates, (b) two serially connected Hadamard gate are together equal to a quantum wire and (c) for controlled Z we can interchange the control qubit and the target qubit in the control-Z gate.

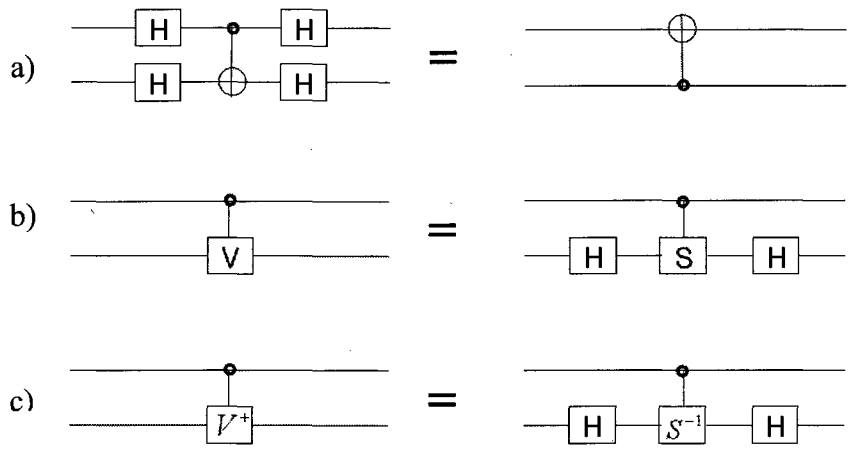
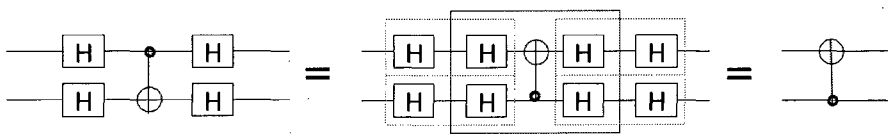
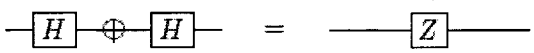


Figure.2.2.3.4 : Identities for Feynman gate surrounded by Hadamard gate and construction of CV and $CV^{\dagger}$ from Hadamard gate , Phase gate(S) and its inverse(S^{-1}).

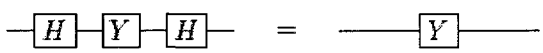


a)



b)

c)



d)

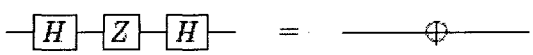


Figure 2.2.3.5: (a) Example of transformation for Feynman gate surrounded by Hadamard gates, (b)Hadamard gate used as serial connection creates Z gate, (c)Y gate surrounded by Hadamard creates Y gate, (d) Z gate surrounded by Hadamard gates creates NOT gate. These rules can be used to prove the correctness of the Grover Algorithm.

2.2.4. Single Qubit Gates

The most frequently used single qubit gates in quantum algorithms are the NOT(N) (*also known as Pauli-X, or X* [Nielsen97]), Hadamard(H), and phase(P) *(also known as S* [Nielsen00]) gates in the vector space spanned by basis vectors in both Dirac and Heisenberg notations from Equation 2.2.4.1:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation 2.2.4.1

These gates are the special cases of the single qubit rotation operations and are implemented by the rotation pulses as shown in Figure 2.2.4.1

$$\text{(a)} \quad N = iR_x(\pi) = i\begin{bmatrix} \cos\left(\frac{\pi}{2}\right) & -i\sin\left(\frac{\pi}{2}\right) \\ -i\sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \text{—[X]—}$$

$$\text{(b)} \quad H = iR_y\left(\frac{\pi}{2}\right)R_z(\pi) = i\begin{bmatrix} \cos\left(\frac{\pi}{4}\right) & -\sin\left(\frac{\pi}{4}\right) \\ \sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{\pi}{4}\right) \end{bmatrix}\begin{bmatrix} e^{-i\frac{\pi}{2}} & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix} = \left(\frac{1}{\sqrt{2}}\right)\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \text{—[H]—}$$

$$\text{(c)} \quad P = e^{i\frac{\phi}{2}}R_z(\phi) = e^{i\frac{\phi}{2}}\begin{bmatrix} e^{-i\frac{\phi}{2}} & 0 \\ 0 & e^{i\frac{\phi}{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} = \text{—[S]—}$$

Figure 2.2.4.1: (a) Calculation of matrix for Pauli X rotation, (b) calculation of matrix for Hadamard gate, (c) Calculation of matrix for S gate.

Therefore, the costs of Gates N and P are said to be 1, and that of H is 2, from the definition of our model quantum computer (see Figure 2.2.4.2). It is worthwhile to note that gates with the same number of input qubits can have and usually have very different costs in practice. The pulse sequence of a gate is not unique in general. It is also worthwhile to note the fact that the N, H and P gates are implemented up to overall phase. We illustrate an example of this fact for the N gate below in Figure 2.2.4.3.

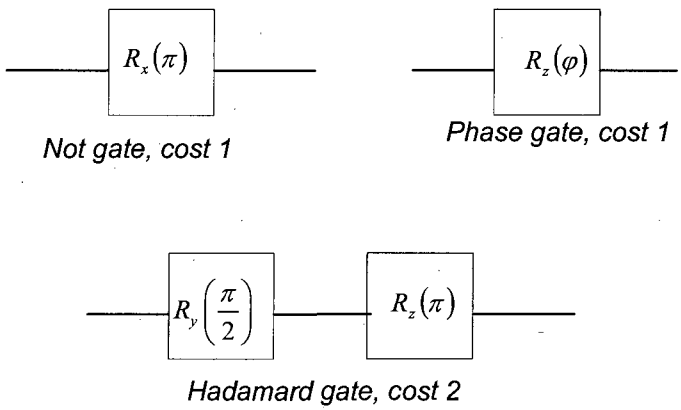


Figure 2.2.4.2: Quantum gates realized on the pulse level, they are decomposed to rotations with respect to axes x, y and z.

Let us denote a NOT gate such that it is correct to overall phase, doing so we have the equations from Figure 2.2.4.3.

$$N = R_x(\pi) = \begin{bmatrix} \cos\left(\frac{\pi}{2}\right) & -i\sin\left(\frac{\pi}{2}\right) \\ -i\sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{bmatrix} = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} = (-i)\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Figure 2.2.4.3: Calculation of unitary matrix for inverter. Illustrates accuracy with phase and relates to the Table 2.2.2. The value of (-i) = (- $\sqrt{-1}$) is the phase that is lost in every quantum measurement.

The concepts of rotations and phase can be illustrated using the Bloch sphere, [Nielsen00].

2.2.5. Two-Qubit Gates

The most frequently used two-qubit gates are the CNOT and SWAP gates. A possible pulse sequences for the CNOT gate is given in Equation 2.2.5.1.

$$CNOT_{ij} = R_{iz}\left(\frac{\pi}{2}\right)R_{jx}\left(\frac{\pi}{2}\right)R_{jy}\left(\frac{\pi}{2}\right)J_{ij}\left(-\frac{\pi}{2}\right)R_{jy}\left(\frac{-\pi}{2}\right)$$

Equation 2.2.5.1: Pulse sequence for CNOT gate (accurate to phase, where i is the target bit).

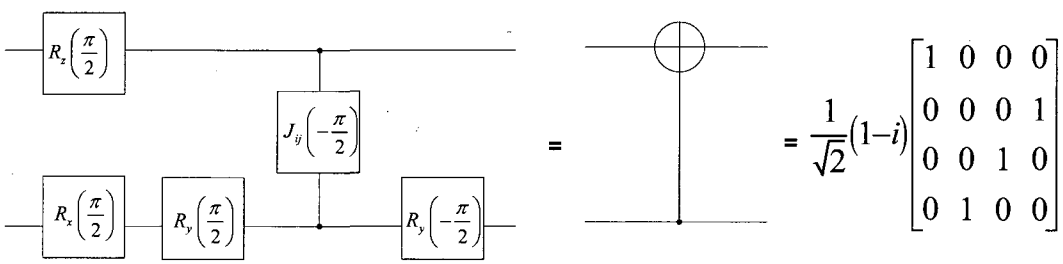


Figure 2.2.5.1: Representation of the CNOT Gate with EXOR up.

Most equations were verified by me using Matlab [MATLAB] and simulation results are presented for the most important circuits in the thesis. Some circuits I simulated also in QuiddPro [QuIDDPro].

Matlab simulation of Figure 2.2.5.1

CNOT =

0.7071 - 0.7071i	*0*	*0*	*0*
0	*0*	*0*	*0.7071 - 0.7071i*
0	*0*	*0.7071 - 0.7071i*	*0*
0	*0.7071 - 0.7071i*	*0*	*0*

Simulation 2.2.5.1

$$CNOT = R_{1y}\left(\frac{\pi}{2}\right)R_{2z}\left(\frac{-\pi}{2}\right)R_{1z}\left(\frac{-\pi}{2}\right)J_{12}\left(\frac{\pi}{2}\right)R_{1y}\left(\frac{-\pi}{2}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

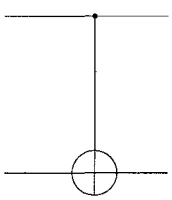


Figure 2.2.5.2: CNOT gate with EXOR down.

Step by step Matlab simulation of Figure 2.2.5.2

R1 =

0.7071 - 0.7071i	*0*	*0*	*0*
0	*0.7071 - 0.7071i*	*0*	*0*
0	*0*	*0.7071 + 0.7071i*	*0*
0	*0*	*0*	*0.7071 + 0.7071i*

R2 =

$$\begin{matrix} 0.7071 & 0 - 0.7071i & 0 & 0 \\ 0 - 0.7071i & 0.7071 & 0 & 0 \\ 0 & 0 & 0.7071 & 0 - 0.7071i \\ 0 & 0 & 0 - 0.7071i & 0.7071 \end{matrix}$$

R3 =

$$\begin{matrix} 0.7071 & -0.7071 & 0 & 0 \\ 0.7071 & 0.7071 & 0 & 0 \\ 0 & 0 & 0.7071 & -0.7071 \\ 0 & 0 & 0.7071 & 0.7071 \end{matrix}$$

R4 =

$$\begin{matrix} 0.7071 + 0.7071i & 0 & 0 & 0 \\ 0 & 0.7071 - 0.7071i & 0 & 0 \\ 0 & 0 & 0.7071 - 0.7071i & 0 \\ 0 & 0 & 0 & 0.7071 + 0.7071i \end{matrix}$$

R5 =

$$\begin{matrix} 0.7071 & 0.7071 & 0 & 0 \\ -0.7071 & 0.7071 & 0 & 0 \\ 0 & 0 & 0.7071 & 0.7071 \\ 0 & 0 & -0.7071 & 0.7071 \end{matrix}$$

CNOT =

0.7071 - 0.7071i	*0 + 0.0000i*	*0*	*0*
0.0000	*.7071 - 0.7071i*	*0*	*0*
0	*0*	*0.0000 - 0.0000i*	*0.7071 - 0.7071i*
0	*0*	*0.7071 - 0.7071i*	*0 + 0.0000i*

Simulation 2.2.5.2: Where R1, R2, R3, R4 and R5 are the Pauli Matrices from Figure 2.2.5.5 and CNOT results from the Equation 2.2.5.1.

In Figure 2.2.5.2 the upper qubit is the control and lower qubit is target respectively. As shown by Equation 2.2.5.1, the cost of a CNOT gate is 5. The circuit corresponding to the equation 2.2.5.3 is shown in Figure 2.2.5.2. Another frequently used controlled gate is the controlled-V where V^2 is equivalent to a NOT gate. The cost of this gate is also 5 because it can be implemented by Equation 2.2.5.3.

$$Controlled - V = R_{2y}\left(\frac{\pi}{2}\right)R_{1z}\left(\frac{\pi}{4}\right)R_{2z}\left(\frac{\pi}{4}\right)J_{12}\left(\frac{-\pi}{4}\right)R_{2y}\left(\frac{-\pi}{2}\right)$$

Equation 2.2.5.3: Pulse sequence for Controlled – V gate (accurate to phase, where i is the target bit).

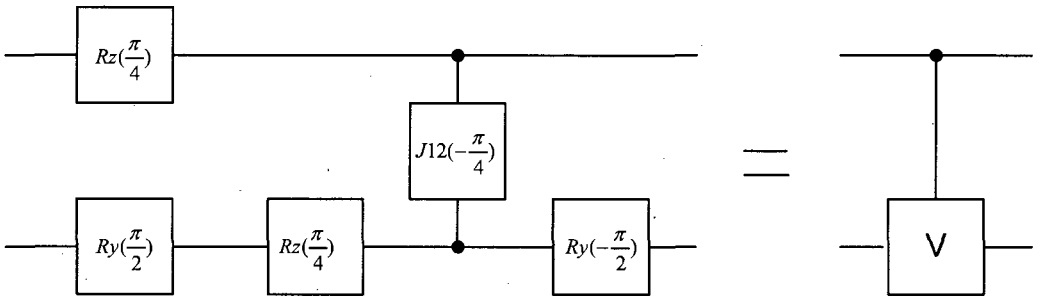


Figure 2.2.5.3: Controlled-V gate realized with 5 pulses.

Once the pulse sequences of the CNOT, controlled-V, and single qubit gates are known, the pulse sequence for the other multi-qubit gates can be obtained if the gate is decomposed to a series of these basic gates.

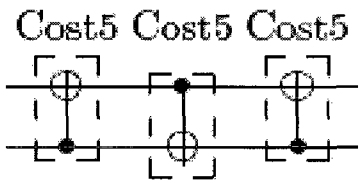


Figure 1.2.5.4: SWAP Gate comprised of 3 CNOT gates. The cost of the SWAP should be then 3×5 = 15 but it is lower thanks to local optimizations based on quantum algebra.

The SWAP gate is decomposed of three CNOT gates as shown in Figure 1.2.5.4. The pulse sequence of the SWAP gate obtained by replacing each CNOT gate (EXOR up) by the sequence in Equation 2.2.5.1 and EXOR down CNOT with sequence from Figure 2.2.5.2 is given in Equation 2.2.5.4. It has cost 15.

$$SWAP = R_{2y}\left(\frac{\pi}{2}\right)R_{1z}\left(\frac{-\pi}{2}\right)R_{2z}\left(\frac{-\pi}{2}\right)J_{12}\left(\frac{\pi}{2}\right)R_{2y}\left(\frac{-\pi}{2}\right)$$

$$\times R_{1y}\left(\frac{\pi}{2}\right)R_{2z}\left(\frac{-\pi}{2}\right)R_{1z}\left(\frac{-\pi}{2}\right)J_{12}\left(\frac{\pi}{2}\right)R_{1y}\left(\frac{-\pi}{2}\right) \qquad \textit{Equation 2.2.5.4}$$

$$\times R_{2y}\left(\frac{\pi}{2}\right)R_{1z}\left(\frac{-\pi}{2}\right)R_{2z}\left(\frac{-\pi}{2}\right)J_{12}\left(\frac{\pi}{2}\right)R_{2y}\left(\frac{-\pi}{2}\right)$$

Using the algorithm from [Lee06] it can be shown that Equation 2.2.5.4 can be reduced to Equation 2.2.5.5 and from Equation 2.2.5.5, the cost of the SWAP gate is 11. The circuit corresponding to Equation 2.2.5.5 is shown in Figure 2.2.5.5.

$$SWAP = R_{2y}\left(\frac{\pi}{2}\right)R_{2z}\left(\frac{-3\pi}{2}\right)R_{1z}\left(\frac{-3\pi}{2}\right)$$

$$\times J_{12}\left(\frac{\pi}{2}\right)R_{2y}\left(\frac{\pi}{2}\right)R_{1y}\left(\frac{-\pi}{2}\right)J_{12}\left(\frac{\pi}{2}\right)R_{1x}\left(\frac{\pi}{2}\right)R_{2x}\left(\frac{-\pi}{2}\right)J_{12}\left(\frac{\pi}{2}\right)R_{2y}\left(\frac{-\pi}{2}\right) \qquad \textit{Equation 2.2.5.5}$$

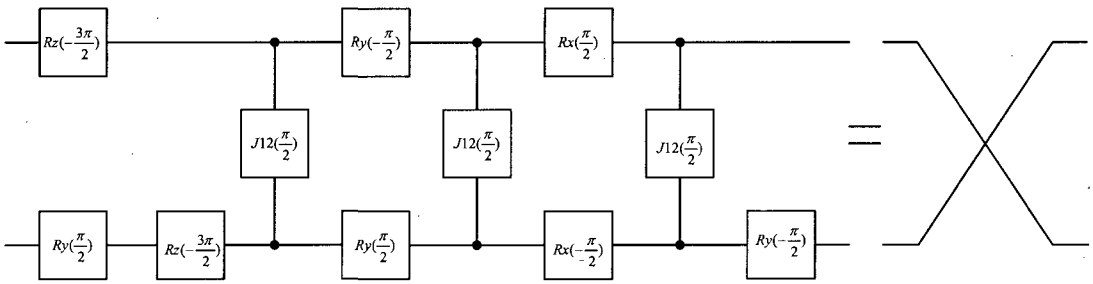


Figure 2.2.5.5: Swap Gate with 11 Pulses.

$$R_x^1(\phi)=\begin{bmatrix} \cos\frac{\phi}{2} & 0 & -i\sin\frac{\phi}{2} & 0 \\ 0 & \cos\frac{\phi}{2} & 0 & -i\sin\frac{\phi}{2} \\ -i\sin\frac{\phi}{2} & 0 & \cos\frac{\phi}{2} & 0 \\ 0 & -i\sin\frac{\phi}{2} & 0 & \cos\frac{\phi}{2} \end{bmatrix} \quad R_x^2(\phi)=\begin{bmatrix} \cos\frac{\phi}{2} & -i\sin\frac{\phi}{2} & 0 & 0 \\ -i\sin\frac{\phi}{2} & \cos\frac{\phi}{2} & 0 & 0 \\ 0 & 0 & \cos\frac{\phi}{2} & -i\sin\frac{\phi}{2} \\ 0 & 0 & -i\sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{bmatrix}$$

$$R_y^1(\phi)=\begin{bmatrix} \cos\frac{\phi}{2} & 0 & -\sin\frac{\phi}{2} & 0 \\ 0 & \cos\frac{\phi}{2} & 0 & -\sin\frac{\phi}{2} \\ \sin\frac{\phi}{2} & 0 & \cos\frac{\phi}{2} & 0 \\ 0 & \sin\frac{\phi}{2} & 0 & \cos\frac{\phi}{2} \end{bmatrix} \quad R_y^2(\phi)=\begin{bmatrix} \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} & 0 & 0 \\ \sin\frac{\phi}{2} & \cos\frac{\phi}{2} & 0 & 0 \\ 0 & 0 & \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} \\ 0 & 0 & \sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{bmatrix}$$

$$R_z^1(\phi)=e^{-i\frac{\phi}{2}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\phi} & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \quad R_z^2(\phi)=e^{-i\frac{\phi}{2}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}$$

$$J_{12}(\phi)=e^{-i\frac{\phi}{2}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & e^{i\phi} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.2.5.6: Two-Qubit Rotation Operations.

The Rotations matrices for two-qubit gates are given in Figure 2.2.5.6. They can be easily used to verify some of the calculations from the thesis.

2.2.6. Three-Qubit Gates

The most frequently used three qubit gates are Toffoli and Fredkin gates, the Miller gate [Miller02] and Peres gate [Peres85] are also used. The circuit diagrams of these four gates are shown in Figure 2.2.6.1. The Peres gate is the cheapest found among those familiar in the universal set of reversible logic gates. It is just like a Toffoli gate but without the last CNOT gate, as shown in Figure 2.2.6.1 (a).

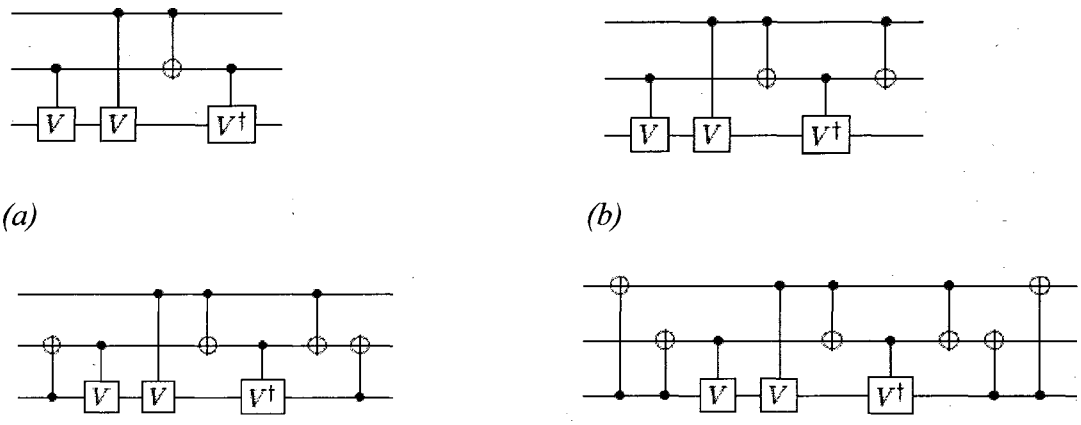


Figure 2.2.6.1: (a)The Peres Gate, (b) The Toffoli Gate, (c)The Fredkin Gate, (d) The Miller Gate

The pulse sequence of the Toffoli gate reduced from the circuit in Figure 2.2.6.1b is composed of 15 pulses and contains 5 interaction terms. However, the equivalent sequence of this gate analyzed by the geometric algebra method presented in [Cory97] is composed of 13 pulses and contains 6 interaction terms. The sequence we listed in Table 2.2.1*Table 2.2.1* for the Toffoli gate is the one with the lower cost. This case indicates that there is at least one quantum circuit for the Toffoli gate more efficient

than shown in Figure 2.2.6.1b, a possibility also exists that the sequences listed in the table can be reduced further. Although the cost of the Toffoli gate given in Table 2.2.1 is lower than the gate shown in Figure 2.2.6.1b, the gate from Figure 2.2.6.1b is practically cheaper than using the method explained in [Lee06]. It is also possible that equivalent sequences can have a different number of interaction terms because $R_{iz}(\pi)R_{jz}(\pi)J_{ij}(\pi)$ is equal to the identity operation. The minimized Peres gate on the level of pulses is shown in Figure 2.2.6.2.

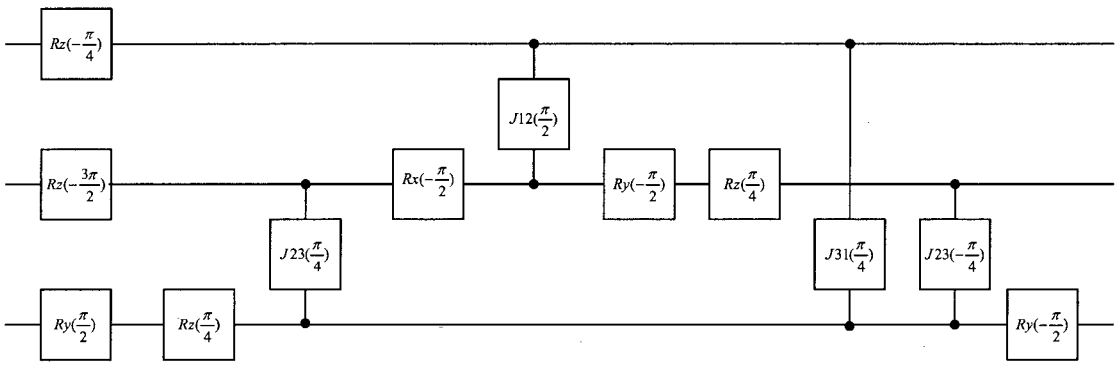


Figure 2.2.6.2: Peres Gate with 12 pulses

The circuit diagram for the "pulse-level" realization of 3 * 3 Toffoli gate is shown in Figure 2.2.6.3. This is perhaps the exact minimum pulse-level realization. This fact has been confirmed by our exhaustive search software. If the search will be completed we will be the first team to prove the cheapest universal gate for quantum computing (most likely Peres gate) and to find the cheapest realization of the Fundamental Toffoli gate.

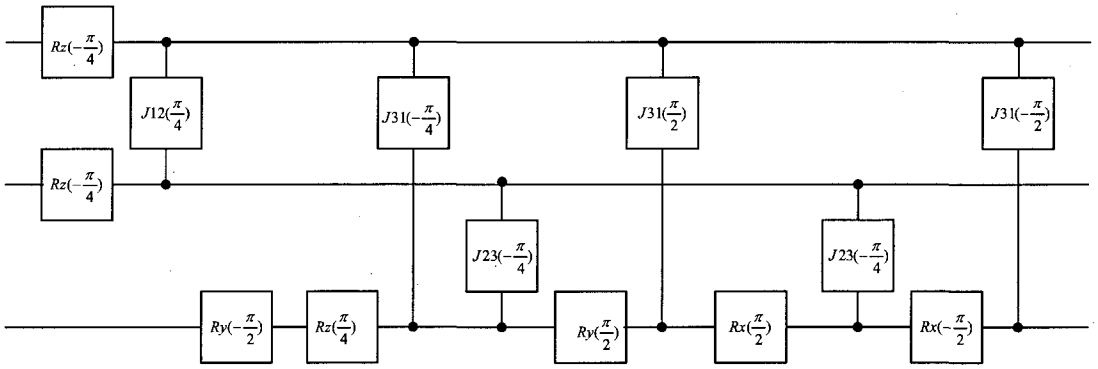


Figure 2.2.6.3: The Toffoli gates with 13 pulses.

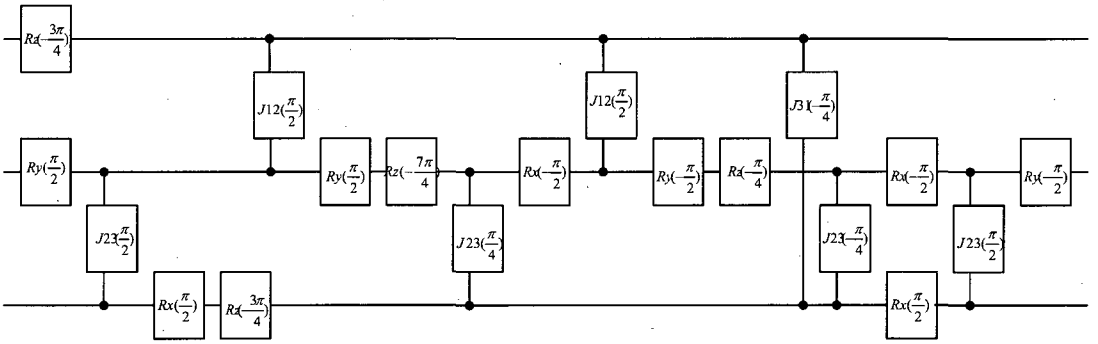


Figure 2.2.6.4: The Fredkin Gate with 19 pulses.

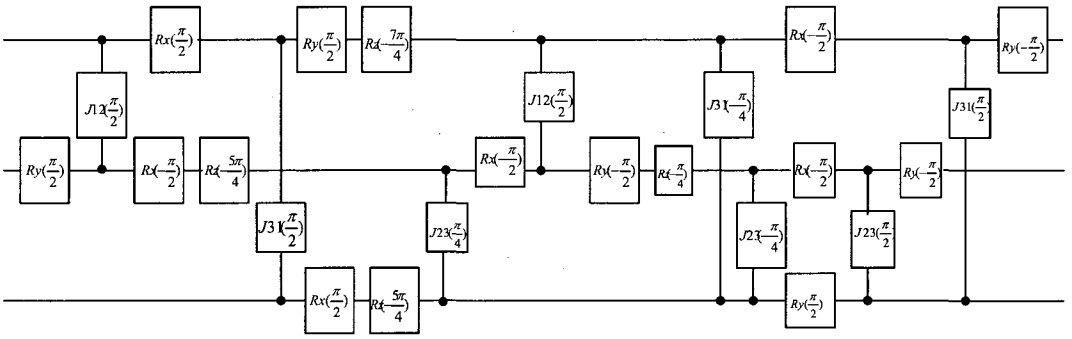


Figure 2.2.6.5: The Miller Gate with 24 pulses.

The circuit for the minimized "pulse-level" Fredkin gate is given in Fig.2.2.6.4. and the circuit for the minimized Miller gate is given in Figure 2.2.6.5.

To explain the fundament of our exhaustive search we will analyze and visualize the Miller gate's pulse level optimization from the Following Equation 2.2.6.1 through Equation 2.2.6.11.

Example 2.2.6.1: Specifically the mathematical analysis is shown in the Equations from 2.2.6.8 through 2.2.6.11.

- NMR Hamiltonian

$$H = \sum_{k} \left(\omega_k I_{kz} + \omega_1(t)[I_{kx} \cos\phi(t) + I_{ky} \sin\phi(t) \right) + \sum_{j,k} \pi J_{jk} 2 I_{jz} I_{kz}$$

Equation 2.2.6.1

- Preferred operations

  Single –qubit operations

  1. Rotation of qubit k by 90° and 180° about the x axis.

$$I_{kx}(\frac{\pi}{2}) \equiv \exp(-i\frac{\pi}{2} I_{kx}).$$ *Equation 2.2.6.2*

$$I_{kx}(\pi) \equiv \exp(-i\pi I_{kx}).$$ *Equation 2.2.6.3*

  2. Rotation of qubit k by 90° and 180° about the y axis.

$$I_{ky}(\frac{\pi}{2}) \equiv \exp(-i\frac{\pi}{2} I_{ky}).$$ *Equation 2.2.6.4*

$$I_{ky}(\pi) \equiv \exp(-i\pi I_{ky}).$$ *Equation 2.2.6.5*

3. Rotation of qubit k by θ about the z axis.

$$I_{kz}(\theta) \equiv \exp(-i\theta I_{kz}).$$ *Equation 2.2.6.6*

Two-qubit operations

1. Rotations of the states of two qubit j and k by θ through the evolution by the coupling term $2 I_{jk} I_{kz}$.

$$J_{jk}(\theta) \equiv \exp(-i\theta\ 2 I_{jk} I_{kz}).$$ *Equation 2.2.6.7*

- Any single-qubit rotation can be accomplished in three steps, known as Euler rotations. The Euler rotations are composed of two z-rotations and one y-rotation. We prefer 90° or 180° y-rotations and the y-rotations in arbitrary angles can be decomposed into two 90° x-rotations and z-rotation.

The original Miller's gate is specified as in Equation 2.2.6.8.

$$\left[I_{1y}\left(\frac{\pi}{2}\right)\ I_{3z}\left(-\frac{\pi}{2}\right) I_{1z}\left(-\frac{\pi}{2}\right)\ J_{13}\ \left(\frac{\pi}{2}\right) I_{1y}\left(-\frac{\pi}{2}\right) \right]$$

$\times$

$$\left[I_{2y}\left(\frac{\pi}{2}\right)\ I_{3z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{2}\right)\ J_{23}\ \left(\frac{\pi}{2}\right) I_{2y}\left(-\frac{\pi}{2}\right) \right]$$

$\times$

$$\left[I_{2y}\left(\frac{\pi}{2}\right)\ I_{1z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{2}\right)\ J_{12}\ \left(\frac{\pi}{2}\right) I_{2y}\left(-\frac{\pi}{2}\right) \right]$$

$\times$

$$\left[I_{3y}\left(\frac{\pi}{2}\right)\ I_{2z}\left(-\frac{\pi}{4}\right) I_{3z}\left(-\frac{\pi}{4}\right)\ J_{23}\ \left(\frac{\pi}{4}\right) I_{3y}\left(-\frac{\pi}{2}\right) \right]$$

$\times$

$$\left[I_{2y}\left(\frac{\pi}{2}\right)\ I_{1z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{2}\right)\ J_{12}\ \left(\frac{\pi}{2}\right) I_{2y}\left(-\frac{\pi}{2}\right) \right]$$

$\times$ $$\left[I_{3y}\left(\frac{\pi}{2}\right)\ I_{1z}\left(\frac{\pi}{4}\right) I_{3z}\left(\frac{\pi}{4}\right)\ J_{13}\ \left(-\frac{\pi}{4}\right) I_{3y}\left(-\frac{\pi}{2}\right) \right]$$

$\times$

$$\left[I_{3y}\left(\frac{\pi}{2}\right)\ I_{2z}\left(\frac{\pi}{4}\right) I_{3z}\left(-\frac{\pi}{4}\right)\ J_{23}\ \left(-\frac{\pi}{4}\right) I_{3y}\left(-\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[I_{2y}\left(\frac{\pi}{2}\right)\ I_{3z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{2}\right)\ J_{23}\ \left(\frac{\pi}{2}\right) I_{2y}\left(-\frac{\pi}{2}\right) \right]$$

$$\times \left[I_{1y}\left(\frac{\pi}{2}\right)\ I_{3z}\left(-\frac{\pi}{2}\right) I_{1z}\left(-\frac{\pi}{2}\right)\ J_{13}\ \left(\frac{\pi}{2}\right) I_{1y}\left(-\frac{\pi}{2}\right) \right]$$

Equation 2.2.6.8

$$= \left[I_{1y}\left(\frac{\pi}{2}\right)\ I_{2y}\left(\frac{\pi}{2}\right) I_{3z}\left(-\frac{\pi}{2}\right)\ I_{1z}\left(-\frac{\pi}{2}\right) J_{13}\ \left(\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[I_{3z}\left(-\frac{\pi}{2}\right)\ I_{2z}\left(-\frac{\pi}{2}\right) J_{23}\ \left(\frac{\pi}{2}\right)\ I_{1y}\left(-\frac{\pi}{2}\right) I_{1z}\left(-\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[I_{2z}\left(-\frac{\pi}{2}\right)\ J_{12}\ \left(\frac{\pi}{2}\right) I_{2y}\left(-\frac{\pi}{2}\right)\ I_{3y}\left(\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{4}\right) \right]$$

$$\times$$

$$\left[I_{3z}\left(-\frac{\pi}{4}\right)\ J_{23}\ \left(\frac{\pi}{4}\right) I_{2y}\left(\frac{\pi}{2}\right)\ I_{1z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{2}\right) \right]$$

$$\times \left[J_{12}\ \left(\frac{\pi}{2}\right)\ I_{2y}\left(-\frac{\pi}{2}\right) I_{1z}\left(\frac{\pi}{4}\right)\ I_{3z}\left(\frac{\pi}{4}\right) J_{13}\ \left(-\frac{\pi}{4}\right) \right]$$

$$\times \left[I_{2z}\left(\frac{\pi}{4}\right) I_{3z}\left(\frac{\pi}{4}\right) J_{23}\left(-\frac{\pi}{4}\right) I_{3y}\left(-\frac{\pi}{2}\right) I_{2y}\left(\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[I_{1y}\left(\frac{\pi}{2}\right) I_{3z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{2}\right) J_{23}\left(\frac{\pi}{2}\right) I_{3z}\left(-\frac{\pi}{2}\right) \right]$$

$$\times \left[I_{1z}\left(\frac{\pi}{2}\right) J_{13}\left(\frac{\pi}{2}\right) I_{2y}\left(-\frac{\pi}{2}\right) I_{1y}\left(-\frac{\pi}{2}\right) \right]$$

Equation 2.2.6.9

$$= \left[I_{2y}\left(\frac{\pi}{2}\right) J_{12}\left(\frac{\pi}{2}\right) I_{2x}\left(-\frac{\pi}{2}\right) J_{1x}\left(\frac{\pi}{2}\right) J_{13}\left(\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[I_{1y}\left(\frac{\pi}{2}\right) I_{1z}\left(-\pi\right) I_{3x}\left(\frac{\pi}{2}\right) I_{3z}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{5\pi}{4}\right) \right]$$

$$\times \left[J_{23}\left(\frac{\pi}{4}\right) I_{2x}\left(-\frac{\pi}{2}\right) I_{1z}\left(-\frac{\pi}{4}\right) J_{12}\left(\frac{\pi}{2}\right) \right]$$

$$\times \left[I_{2y}\left(\frac{\pi}{2}\right) J_{13}\left(-\frac{\pi}{4}\right) I_{2z}\left(\frac{\pi}{4}\right) I_{3z}\left(\frac{\pi}{4}\right) J_{23}\left(-\frac{\pi}{4}\right) \right]$$

$$\times$$

$$\left[I_{2z}\left(-\frac{\pi}{2}\right) I_{3z}\left(-\pi\right) I_{3y}\left(\frac{\pi}{2}\right) I_{2x}\left(-\frac{\pi}{2}\right) I_{1z}\left(-\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[I_{1x}\left(-\frac{\pi}{2}\right)\ J_{23}\ \left(\frac{\pi}{2}\right) J_{13}\ \left(\frac{\pi}{2}\right)\ I_{2y}\left(-\frac{\pi}{2}\right) I_{1y}\left(-\frac{\pi}{2}\right) \right]$$

Equation 2.2.6.10

$$=$$

$$\left[I_{2y}\left(\frac{\pi}{2}\right)\ J_{12}\ \left(\frac{\pi}{2}\right) I_{2x}\left(-\frac{\pi}{2}\right)\ I_{2z}\left(-\frac{5\pi}{4}\right) I_{1x}\left(\frac{\pi}{2}\right) \right]$$

$$\times$$

$$\left[J_{31}\ \left(\frac{\pi}{2}\right)\ I_{1y}\left(\frac{\pi}{2}\right) I_{1z}\left(-\frac{7\pi}{4}\right)\ I_{3x}\left(\frac{\pi}{2}\right) I_{3z}\left(-\frac{5\pi}{4}\right) \right]$$

$$\times$$

$$\left[J_{23}\ \left(\frac{\pi}{4}\right)\ I_{2x}\left(-\frac{\pi}{2}\right) J_{12}\ \left(\frac{\pi}{2}\right)\ I_{2y}\left(-\frac{\pi}{2}\right) I_{2z}\left(-\frac{\pi}{4}\right) \right]$$

$$\times$$

$$\left[J_{31}\ \left(-\frac{\pi}{4}\right)\ I_{23}\ \left(-\frac{\pi}{4}\right) I_{3y}\left(\frac{\pi}{2}\right)\ I_{2x}\left(-\frac{\pi}{2}\right) I_{1x}\left(-\frac{\pi}{2}\right) \right]$$

$$\times \left[I_{23}\ \left(\frac{\pi}{2}\right)\ I_{2y}\left(-\frac{\pi}{2}\right) J_{31}\ \left(\frac{\pi}{2}\right)\ I_{1y}\left(-\frac{\pi}{2}\right) \right]$$

Equation 2.2.6.11

The subsequent equations show one branch of the search tree which leads to the supposedly minimal solution that we dispose at this point.

Example 2.2.6.5.2:

Now, we graphically visualize the Miller gate's pulse level optimization from the following Figure 2.2.6.6 through Equation 2.2.6.8 as below:

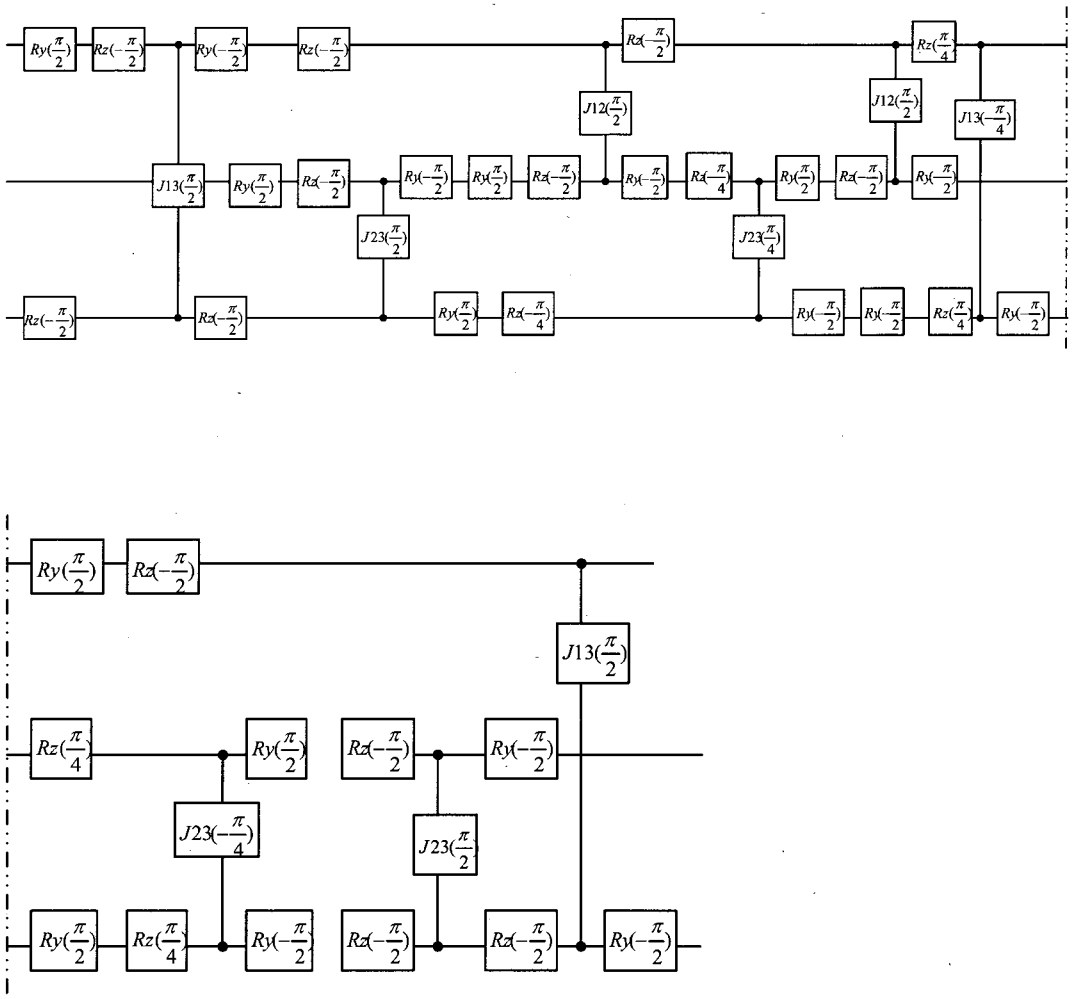


Figure 2.2.6.6: Miller Gate realized with 45 pulses from Equation 2.2.6.8.

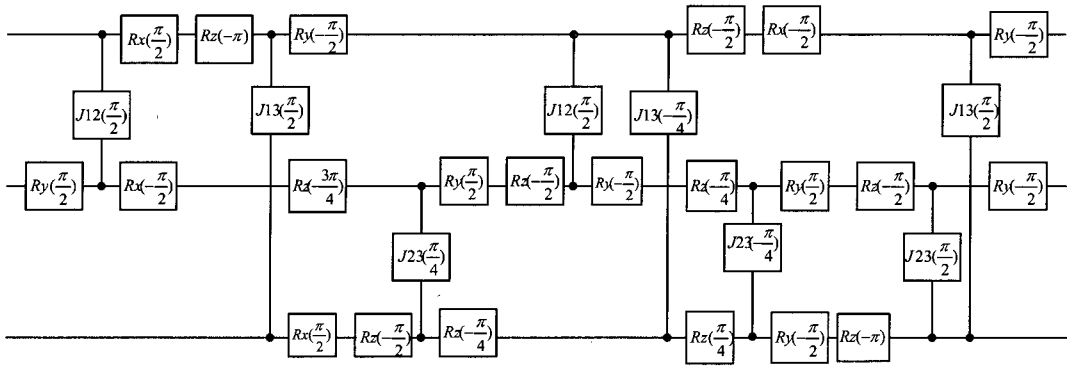


Figure 2.2.6.7: Miller Gate realized with 30 pulses from Equation 2.2.6.10.

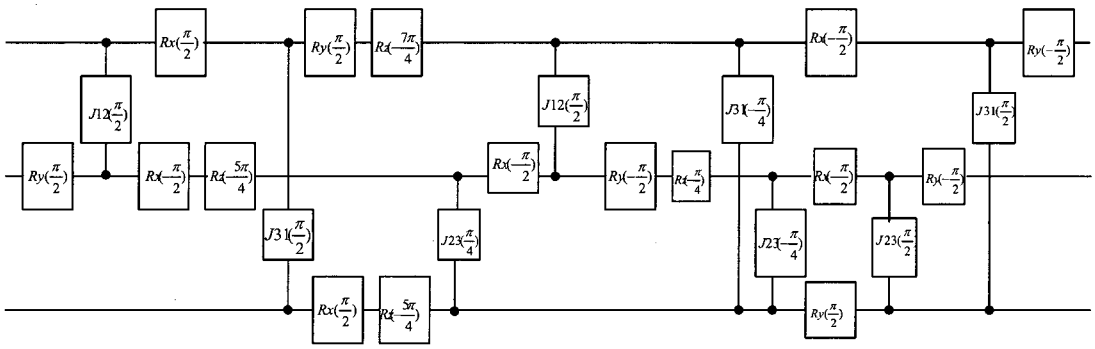


Figure 2.2.6.8: Optimal Miller Gate Realized with 24 pulses from Equation 2.2.6.11.

These figures can be compared with the macro-level specification of the Miller gate using 2×2 quantum gates from Figure 2.2.6.1d.

2.2.7. Large gates and gates for the "neighbor-only" technology

Example 2.2.7.1:

In some technologies such as "Linear ion trap" [Leibfried03] every qubit can communicate only with its neighbors above and below; this increases the cost of gates.

If we have a wire that is "going through" the Feynman gate (Figure 2.2.7.1b), what should we do? We have to create a sequence of Feynman gates realizing SWAPs (Figure 2.2.7.1). The realization of Toffoli gate itself in the "neighbor-only" technology is shown in Figure 2.2.7.2. Again the SWAP gates should be transformed as in Figure 2.2.7.1a.

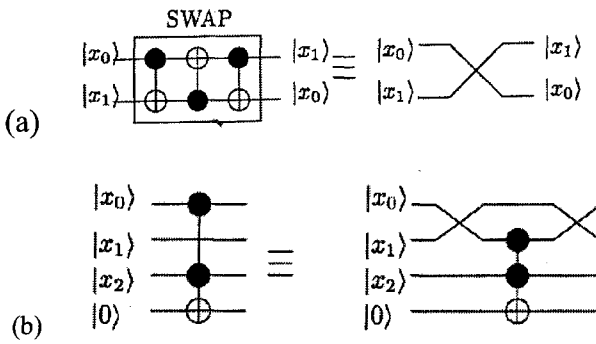


*Figure 2.2.7.1: Transforming a 3*3 Toffoli gate with qubit X_1 going through. (a) the SWAP gate, (b) the transformation of the Toffoli gate by surrounding it with two SWAP gates. Each of these SWAP gates is next transformed as in Figure 2.2.7.1a.*

Example 2.2.7.2:

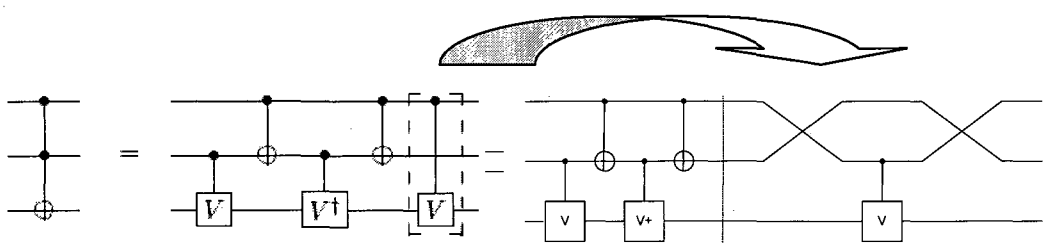


Figure 2.2.7.2: Realization of Toffoli gate in the technology that allows interactions only between neighbor qubits.

Example 2.2.8.3:

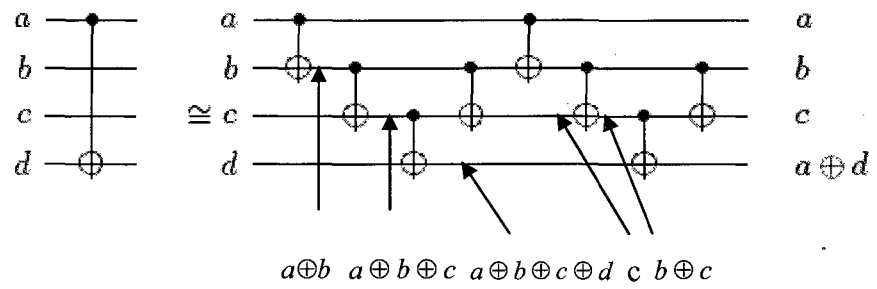


Figure 2.2.7.3: Transformation of "big CNOT" gate in the "neighbors only" quantum Technology. This is a Feynman gate with two qubit wires "going through" it.

A CNOT gate with many qubit wires "going through" can be realized as shown in Figure.2.2.7.3. Please note the Boolean equations used in the verification process. As we see from these simple examples, the "neighbor-only" technologies increase very substantially the costs of gates and circuits satisfying their linearity constraint. These effects were entirely not taken into account by the previous researchers thus the claimed by them "minimal circuits" are in fact very far from the minimum when one calculates their costs on "pulse level" rather than "abstract mathematical gate level" (like n-input Toffoli). This is why we create affine gates and similar concepts in next chapters, and why in some variants we take the "neighbor only" constraint of linear Ion-Trap.

2.3. Realization of Fredkin Gate Using Cellular Automata

2.3.1. Non-quantum Realization of Reversible Binary Gates.

The permutative gates such as Toffoli, Peres or Fredkin can be built in the technologies other than NMR. The synthesis methods that will be presented in all next chapters are general and they operate on the level of permutative gates. It means, that when the gate is already designed using lower level primitives, it is treated as an entity. Therefore, all next methods will not depend on the internal realization of the gate and the gate may be either reversible non-quantum, or quantum. Only when taking the cost of gates (circuits) into account we will refer to the quantumness of the lower-level realizations.

Please note thus, that the gates themselves, such as Fredkin, Feynman and Toffoli can be also build in non-quantum technologies such as Optical or Nano-technologies. Here it will be illustrated how I was able to build the Fredkin gate using a new non-partitioned, Moore-neighborhood, 2-dimensional cellular automata. The background knowledge of cellular automata can be found in [Buller03a, Fredkin03, Hanson93, Ilachinski01, Kari94, Kari96, Margolus03, Morita94, Wolframe02, Wireworld1, vonNeumann66, Toffoli90]. Such automata are models for several nano-technologies and are universal models of computing on micro-level, equivalent to Turing Machines. This method can be used for the modeling of Boolean logic gates and general circuit construction. Various types of signals can be modeled as well as various stable

architectures that use oscillating and quasi-oscillating elements. These cellular automata are capable of modeling various circuit's functions, all classical logic gates, and can implement Fredkin, Toffoli, and other reversible gates, thus showing universal computation. The CAs can achieve given input/output requirements by cellular signal/architecture/oscillator interactions or by signal collisions. The model has the ability to construct what Fredkin calls the Arbitrary Machines [Fredkin03]. My model meets many of the requirements for a 2-dimensional universal construction as outlined by Miller and Fredkin [Miller97]. Two methods of gate construction using the Fredkin reversible gate will be given below.

Cells are simple identical information processing machines and a cellular automaton is an iterative array of cells where each cell can communicate with neighboring cells. These cells can change from one state to another as a function of the state of the cell and states of its neighbors at discrete moments of time. Thus the collection of cells is characterized by some type of behavior on a global basis. Here we will introduce a new non-partitioned, Moore neighborhood 2-dimensional Reversible Cellular Automata (RCA) which is capable of modeling various functions, all classical logic gates, and can implement Fredkin, Toffoli, and universal reversible computation. The goal of our investigation was to determine how simple the individual cells should be for the global behavior to achieve some specified criterion of complexity, like the ability to perform a computation or to reproduce some pattern. The physical relevance of reversibility in computation and a discussion of time/space trade-offs involved in

reversible computation are introduced in [Bennett89, Morita94, Toffoli90]. In addition we refer the reader for the brief history, aims, uses, decidability and large bibliography of RCA field [Bennett82, Morita94, Kari94]. The 2-D CA shown by Banks in [Banks71] is known to be universal constructor and also Cellular automata capable of universal computation based on BBM (Biliard Ball Model) [Margolus87, Margolus03, Fredkin82]. In [Miller97] Miller and Fredkin described a two-state three-dimensional RCA capable of universal computation.

2.3.2. The Builder CA

The class of 2-dimensional cellular automata rules includes a subset family of generation or history rules. The distinctive character of these rules is that cells besides having the binary property of being in "off" or "on" states, also can have quasi-on/quasi-off, decay states called "histories". These states do not interact with other "living cells" in the array, except in an inhibitory function. No new growth can occur in cells that are in "history states", nor do history state cells incite new growth in their Moore neighborhoods. They are in effect "dead" cells in the array, removed from the general computation. As a convention we denote cells in the on-state as having value 1, cells in the off-state as having value 0 and history or decay cells as having values n-1, n-2, n-3, n-4..... The general effect of history cells in a Moore neighborhood is to vector growth in the directions away from the history cells.

The introduced above global property of history rules leads to a great deal of flexibility and resource in modeling signals and signal processes. The history rule: 0345/26/6 meaning that cells survive from one clock cycle to another if their Moore neighborhood has either 0, 3, 4, or 5 neighboring cells in the on state; that there is cell growth if the Moore cell count is either 2 or 6 live cells; and that all cells have 6 possible states: 0 (off) 1 (on), and 4 intermediary decay states. This particular one of the history or generation rules, defined as **Builder,** 0345/26/6, means that cells survive from one clock cycle to another if their Moore neighborhood has either 0, 3, 4, or 5 neighboring cells in the "on" state. It means that there is cells' growth if the Moore cell count is either 2 or 6 live cells; and that all cells have 6 possible states: 0 (off) 1 (on), and 4 intermediary decay states. It can be used to model all classical logic gates (NAND, EXOR, NOT, AND, OR, NOR, etc), and the classical versions of many quantum gates (Toffoli, Fredkin, Swap, CNOT, etc). It can also model signals as discrete impulses, waves, etc. Signals can be made to travel free or move as multi-valued pulses along conducting elements or within channels. Signals can be deflected in x or y axis, reflected, modulated, damped, delayed, accelerated, stored, multiplied or deleted. Signal streams can be manipulated in various ways, including pulsing, un-pulsing, merging, splitting, shifting, redirecting, and selective cancellations. The rule is isotropic, so all functions are unaffected by interchanging x and y-axes and 90 degree rotations. There are 4 decay states that cells may have between the "on" and "off" states. All growth and signal propagation is at Moore speed which is defined as one contiguous cell per cycle. Here a signal is a moving, self-renewing group of live

cells that propagates in the array. A P**** clock is an oscillator or quasi-oscillator that emits an unending stream of signals. The P value is the cycle time between signal emissions. Any integral value period is constructible in **Builder** and this rule is based on the Moore neighborhood.

Aiding these operations, the rule supports the construction of cellular devices that are oscillatory and these oscillations can be made of any integral period; even, odd and prime. Such devices can be used to interact with individual signals or signal streams to generate synchronized circuit networks of great complexity. The rule also supports cellular "guns" or clocks that emit signals in any integral periodicity. This allows the designer to choose clock speeds for individual circuits. The P0019 block oscillator is shown in Figure 2.3.2.1.

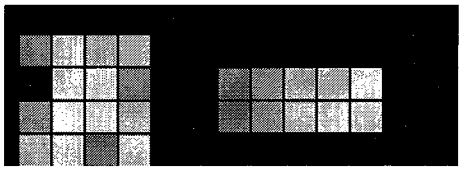


Figure 2.3.2.1: The P0019 block oscillator

The rule supports stable architectures that can serve as conducting elements for signals or as gates for signal interaction. Stable architectures work in any clock cycle. Architectures, like signals, have fundamental properties based on their geometries. They are either traversible by a signal or not, and the transit time must be either an odd or even natural number. These two properties are crucial in their effects on signals interacting with them. A second topological property is that of connectivity

and non-connectivity: traversible structures may be of either type and although connectivity does not effect either transit time or traversibility, it has a key effect on signal interactions.

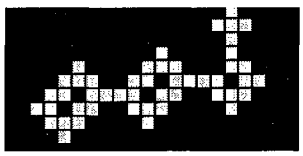


Figure 2.3.2.2: The T25 junction

Any architecture that is traversable and many non-traversable architectures can be made into a quasi-oscillator. This property adds another resource to the methods of working with signals. Because of the multiple possibilities of architectural geometries, signals can interact with these architectures in many different ways. The optimized EXOR gate realization in Builder is an example of a gate based on a quasi-oscillator.

Signals influencing on architecture can destabilize the architectures or preserve them and be damped or destabilized themselves, or preserve the architectures and be channeled into useable outputs. It is this latter class of interactions that are used to build the logic gates and memories. A Fredkin gate realization in Builder employs a property of a specific type of traversable connected architecture surface that allows impinging signals to break into two separate forms of conducting pulses. These two types of pulses have algebra of interaction with each other on the surface of the architecture that allows them to either reflect off each other, cancel each other both, cancel one or the other, pass through each other or swap with each other.

Architectures can also take two or more signals and make them interact with each other to generate distinctive outputs. Another realization of a Fredkin gate in this rule utilizes this property to make the control signal alter the path of the target signal. Signals can be modulated from the standard minimum signal to forms that have parity values of various degrees of complexity. These more complex signals also interact with architectures and oscillators in very distinctive ways. Signals can also be modeled as waves that are continuous and are active along their entire wavefront. These too can be made to interact with oscillators and stable architectures to generate useable outputs.

The 0345/26/6 rule also supports signals that move at one/half Moore speed and 2/3 Moore speed. In addition one can observe that on conducting elements or travelling freely elements, their signal pulses can be accelerated or delayed from the Moore speed constant. Besides interacting with oscillators, quasi-oscillators, or stable architectures, signals can interact with each other, and collision-based computation can be modeled in the Builder rule. Since the rule supports signal forms that move at different Moore speeds, it is possible to use streams of differing speed signals to interact with each other to various effects. These effects in my designs include the following: deflection, remote cell placement, remote oscillator creation, remote clock creation, and remote quasi-oscillator creation. These latter structures can also serve as memories based on their cycle periods. With these properties, the rule meets many of the criteria for the Wolfram's "universal construction" capability [Wolfram02].

Generally speaking, for any given desired signal output in this particular cellular domain, there exists a minimum of four distinct approaches to realizing it, given a specific input. The approaches are:

- stable architectures interacting with signals;
- oscillator and quasi-oscillators interacting with signals;
- mixtures of stable architectures and oscillating elements interacting with signals;
- signals interacting with each other via collision or Moore neighborhood approach distances.

This gives the designer a great deal of leeway in handling difficult modeling tasks. There have been multiple uses of two-dimensional (2D) cellular automata employing non-partitioned arrays with Moore neighborhoods to model circuit functions. J.H. Conway's famous **Life** rule [Wolfram02] uses intersecting streams of "gliders" to achieve selective cancellations and hence binary signal streams. The **Wireworld** rule of B. Silverman [Wireworld1] employs 4-state cells to generate conducting wires and signals traversing those wires. The famous **Billiard Ball Machine** of E. Fredkin [Fredkin82] uses signal collision and reflection based methods. Interestingly **Builder** is capable of emulating some features of all these approaches. Colliding stream with selective cancellations can be modeled as "moving signals through channels". For the purpose of this section, we use alternative methods not illustrated in these other projects, and not discussed in this dissertation for brevity.

2.3.3. The Fredkin gate – one method of modeling using stable architectures only

For the distinctive Fredkin output, we need three input signals: one control signal that passes directly to output, and two target signals. For the purpose of this model, standard signals moving at Moore speed are used. The Fredkin gate's fundamental to the property of reversibility from my work is achieved by flipping the values of the A and B signals under the influence of the control signal. This allows for the recovery of the original input when two Fredkin gates are cascaded.

In **Builder** the setup is fairly straightforward. We set up 3 clocks at P0045 to generate signals streams for the control signal and two target signals A and B. The clocks themselves are quasi-oscillators that have P0045 periodicity (Figure 2.3.3.1). This particular periodicity was chosen because of the clearance traverse time of pulses on one of the gate elements and signal path crossing timing issues.

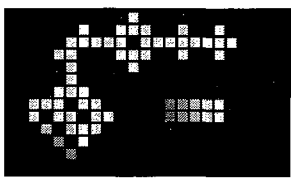


Figure 2.3.3.1: P0045 Clock

Once the clocks are constructed and positioned, then the architecture that doubles the control stream is constructed and placed so that the free signals in the C control stream

interact with this architecture. The purpose of the design is to ensure that the control signal reaches the output and is still able to interact with the target signal(s). The control signal in this method is run through two signal doublers -- architectures which have the ability to double the signals traversing them -- so as to create three control streams: one to output, one to an architecture where it meets the A stream and one to an architecture where it meets the B stream (see Figure 2.3.3.2).

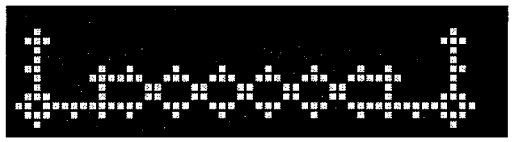


Figure2.3.3.2: The signal doubler

The A and B streams are simultaneously routed to two architectures, one for each, that are designed to accept the free signals, convert them to bifurcated conducted pulses that traverse the architecture's odd-integral value surface and emerge again as free signals traveling to the output.

The Control stream is routed in both of these architectures where it also is converted and bifurcated and interacts with the signals arriving from the A and B clocks. There the four pulses (two of the C and 2 of the A or B) interact. One pulse each of the C and the target pulses cancel each other and the two remaining pulses combine to emerge as a free signal. The distinction is that this free signal emerges at a shifted focus on the structure and thence travels a different path then the uncontrolled A and B exit points and paths. This is one of the key features of the gate, the ability of an architectural element to shift stream exit paths given control signal inputs.

The design now has five possible output streams: A, B, C, CA and CB. The remaining design is to take the A, B and CA and CB paths, direct them to another architecture that will handle them differentially and generate the characteristic Fredkin outputs. We want A to go to A' output, B to go to B' output, CA to go to B' output and CB to go to A' output. C is already routed direct to C' and its doubles have been cancelled out at the junction architectures.

The architecture chosen for the task is one that has the ability to receive incoming signals and convert them to two forms: a single layer pulse and a double-layer pulse. Both types of pulses have characteristic architectures that allow them to emerge from the structure as free signals moving to the outputs. One end of the architecture is constructed to allow only single layer signals to exit, the other end will allow only double layer signals. The A signal is converted to double layer form, the B signal to single layer form. When the control signal is present at the junction device, CA and CB signals arrive and are given opposite values. The CA signal generates a single layer form on the architecture and the CB signal generates a double layer form. The task of this architecture is to route the A, B, CA, and CB streams to the appropriate outputs, achieve the necessary cancellations and signal pass-throughs, and be able to handle any combination of incoming signals moving along four paths at full traffic loads.

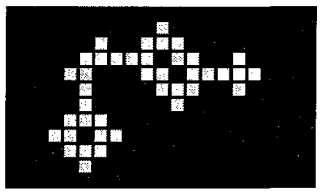


Figure 2.3.3.3: The surface of the interleaver.

The last point, is the hardest to accomplish with this type of architecture, as signal paths have x or y axis displacement causing transit times to vary. The fastest clock cycle this approach can run at is P0041. This is caused by the necessity of the conducted signals clearing the reception area of the architecture before the next wave of signals arrives.

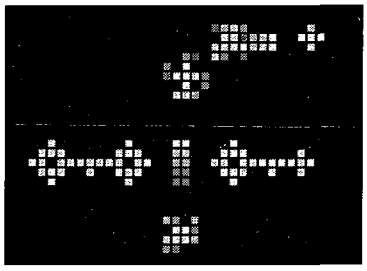


Figure 2.3.3.4: The signal tripler

The last details of the realized Fredkin gate are timing issues at the outputs and x/y displacement of the outputs so cascading can be achieved. Since different path distances are involved, C, A, B, CA and CB signals arrive at the outputs at differing times. We use the delay architecture to synchronize the inputs and outputs. The bulk of the gate is tied up in such architectures and the outcome is that the gate transit time as a whole is adversely affected. In the design the final gate has a 400+cycle transit

time which seems unacceptably high. For this reason and the slow clock time, another approach seems wise.

2.3.4. The Fredkin gate – a faster method utilizing both stable architecture and oscillating elements

The desire here is to preserve a high P0019 clock rate and minimize the transit time. An entirely new approach is employed that removes the interleaver structure and relies solely on junctions. The setup is similar to the previous one with a few minor changes. P0019 clocks replace the P0045 clocks. The C control stream is run through one signal tripler. This is an architecture constructed to accept a signal traveling on a given axis, send copies in positive and negative directions in the other axis and lastly let the original signal emerge continuing along its original direction. An x axis control signal then generates two y-axis signals moving in opposite directions to each other, and then continues along the x-axis to its output point.

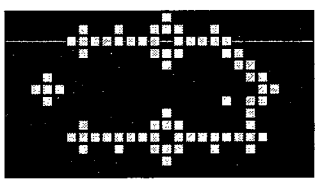
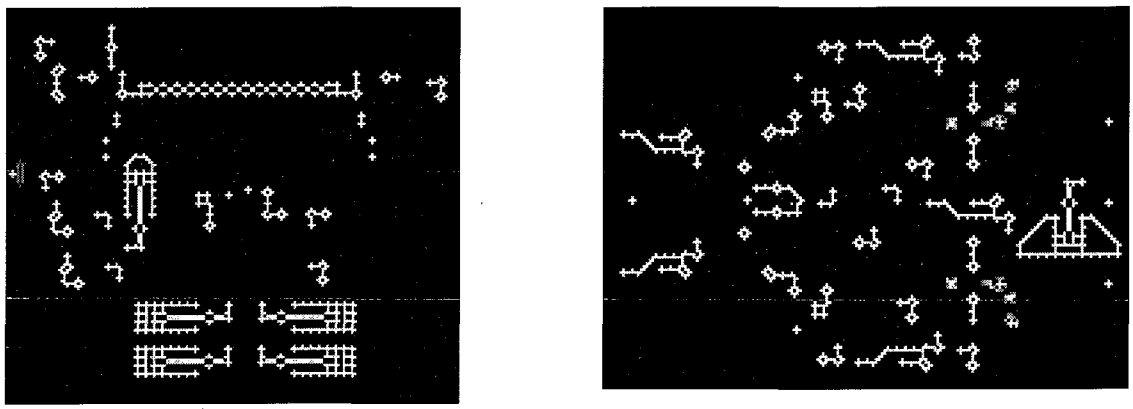


Figure 2.3.4.1: Illustration of OR gate

The A and B streams, as in the other model, go to junctions where they can encounter the control signal and be modified in their exit paths. With this idea, the design approaches are similar: taking A and B signals and giving them four possible paths to the outputs A' and B'.



(a) (b)

Figure 2.3.4.2: (a) The Fredkin v1 gate in Builder and (b) Fredkin Gate v2 in Builder

The change in this faster method is that the interleaver architecture is left out and we achieve the same ends by arranging that A and CB streams meet at a variety of OR gates. At another OR gate the B and CA streams meet. The OR gates used have the property of being able to accept signals coming in along different paths and converting them to synchronized outputs moving at one single path. To do this, the OR gate has two small architectures that take incoming signals, cancel out one of the two conducted pulses generated, conduct the remaining pulse down to a gap where it too cancels out. Passing through the gap continuously are free signals generated by a P0019 clock. These signals act to inhibit signal firing at another P0019 clock. When the conducted A, B, CB, and CA signals are conducted into this gap at the proper timing, they delete out the free signals. This allows the inhibited clock to fire a signal toward the output. Thus the synchronization of the P0019 basic cycle is effected although we have differing path lengths for the four signal streams. The two OR gates

take the four signal streams and generate two output streams going to A' and B'. This completes the construction of the gate.

Both methods realize the Fredkin gate's input/output table, minimize garbage signals and can be used as design elements within larger circuits. Both methods rely heavily on architectures that deflect, and delay free signals so as to ensure timing constraints. Using similar methods, the Toffoli, Swap, Simple Majority, and CNOT gates have also been constructed and can be demonstrated on software during the dissertation defense. Toffoli gate of any size can be built from Fredkin and CNOT gates. Therefore, all developed by me synthesis methods from next chapters are applicable to cellular automata and thus all physical models described by cellular automata [Toffoli90, Fredkin82, Fredkin03, Wolfram02].

2.3.5. Conclusions on my Cellular Automata designs.

The **Builder** rule is highly flexible in the resources available to the designer. All classical gates can be constructed. Most gates exist within multiple design types that have varying throughput times and cell counts. The optimized XOR gate, for instance, is almost 90% smaller than its biggest cousins. Clocks can be optimized too: the first P008 clock designed in this rule had a cell count of over 2150 cells, three increasingly optimized versions reduced the cell count to 79 only!

The rule's weakness is that being path-dependent, most of the gates rely heavily on routing and delay elements that significantly impair the transit times for signals to clear. The bulk of a circuit constructed with this rule is tied up in such elements. The longest path determines the speed of the gate and even the longest paths are made longer by the necessity for delay elements to achieve output timing requirements.

Counterbalancing this, a very rich array of inter-signal collision effects is possible, and various architectures can be made on these principles: from extreme simplicity to baroque intricacy. Since the rule supports periodicity of any integral value, a large number of interactions based on differential timing are available. The Fredkin gates presented here are probably not the optimum in cell count, cycle speed, or throw-put time. But they do point out what is possible.

2.4. Conclusion on Technologies.

All gates that will be used in next chapters are based on quantum gates from this chapter. The quantum costs that we developed and illustrated in this chapter will be used in the entire thesis.

CHAPTER 3

The AND EXOR Logic

3.1. The AND/EXOR logic to synthesize quantum circuits on level of permutative gates

3.1.1. The choice of logic synthesis methods for quantum circuits

In section 2.2 we explained about the lowest level synthesis of quantum circuits in an existing technology and in section 2.3 the synthesis of reversible (permutative) circuits in a general cellular automata model proposed for various nanotechnologies [Wolfram02]. The question appears – *"how to specify permutative circuits on gate level in a way convenient to oracle designer and next how to convert this (higher level, more abstract) specification into an optimized circuit with these permutative gates."* This is one of fundamental questions of this thesis.

All methods from next chapters that will optimize circuits on level of Toffoli, Peres, Feynman and Fredkin gates are good for arbitrary technology used to realize the gates themselves. They can be thus used for **any realization** of Peres or Fredkin gates, including those from the sections 2.2 and 2.3 below. There are however, two ways of using the oracle. The classical oracle obtains all its inputs sequentially, this can be applied with the reversible circuit in any technology. The quantum oracle obtains from the input-level vector of Hadamard gates the superposition of all states in parallel, and thus superposed signals are transmitted to the gates inside the oracle. Thus, in case of

quantum oracles, only the quantum realizations of permutative (reversible) circuits (gates) are allowed.

From now on we will concentrate therefore only on NMR technology but we hope the reader understands that our circuit synthesis methods apply to all realizations of permutative circuits, however with different methods of cost calculation. Various methods have been proposed for permutative circuit synthesis and optimization, of which the historically first and so far the most advanced are methods of evolutionary algorithms, specifically Genetic Algorithms and Genetic Programming. It is well known that Genetic Algorithm (GA) [Goldberg89, Holland92] and Genetic Programming (GP) [Koza92, Koza94, Koza99] techniques provide means for applying the theory of Darwinian evolution within an artificial system. The GA is a system that evolves problem parameters directly; the GP evolves programs for problem solution. Through a process of emergent intelligence, the GA/GP formulates engineering solutions based on an accumulated knowledge of the problem and the merit of potential solutions. In recent years the Genetic Algorithm and Genetic Program, as the machine learning techniques, have been successfully applied to a wide range of engineering problems and were the main methods used in other research groups that work on quantum circuits design [Rubinstein01, Spector99, Willimans99] and also in our research group for quantum circuits synthesis [Lukac02, Lukac02a, Lukac03, Lukac05, Khan03, Khan04, Giesecke07]. However, these methods brought only

limited success to the design of circuits, as they use knowledge insufficiently. Past experience has shown that the GA application to logic minimization has serious limitations of size, computation time, and solution optimality [Dill97b, Dill01]. A question arises, if once the quantum computers are developed, will it be a good idea to use GA and GP on them? Or rather use a general–purpose processor with software GA algorithm to synthesize the quantum circuits? We cannot find anything in a quantum computer that would make such a computer to be principally superior to a standard computer with respect to realizing classical Darwinian evolutionary algorithms (of course quantum computers will have technological advantages such as speed and low power, but I mean here the fundamental algorithm complexity). We can, however, still make use of quantum computer general speed-up in Grover-like algorithms to adapt standard GA-like algorithm to quantum computers. We can still use the general metaphor of evolution through chromosomes, genotypes, phenotypes and survival of the fittest. Another method to be tried is the exhaustive search – again, useful in the first phase of research and well-adaptable to Grover-like algorithms. Yet other methods are heuristic search methods which use knowledge – the so-called structured or informed search approaches. Before we discuss our algorithms and hardware for synthesis, we will systematically introduce the background, this time not the technology level of circuits but the logic level of circuits will be discussed. Now that after reading chapter 2 the reader is more familiar with the basic underlying technology

we can be more specific than in Chapter 1 and we will try to motivate our use of gates, structures, circuit specifications and algorithms.

3.1.2. Reed-Muller Logic, Permutative Logic and Quantum Computing

Most of the current CAD tools in classical computing utilize AND-OR design implementations for both logic synthesis and minimization, both for two-level and multi-level design. These minimization tools are used, also because of historical reasons, in the development of standard digital systems and can be potentially adapted to quantum circuits. However, the fundamental permutative gates in quantum logic are CNOT (Controlled NOT) which uses EXOR gate, Toffoli (which uses double-controlled NOT or C = ab $\oplus$ c function), Fredkin, Peres and generalized Toffoli, like abcde...n $\oplus$ m. As discussed in section 2.2, these gates are internally build from Controlled-V (Controlled Square root of NOT) and its adjoint gate Controlled-$V^{\dagger}$ [Yang05, Yang05a, Yang05b]. The basic classical logic components of quantum gates and quantum design are therefore not the AND and OR operators but the AND and EXOR operators, which means the CV, CV^{+} gates on the lower level level of description. The algebra of EXOR and controlled circuits (with commutative operations like *(a $\oplus$ c)* and non-commutative operations like *(a CONTROL c)* is not similar to AND/OR/NOT Boolean logic and all respective methods based on Boolean laws (like finding prime implicants, graph coloring to minimize the cover of minterms

with prime implicants or unate/binate covering approaches for two or more –level circuits optimization). In contrast to the classical CMOS logic where the realization of the EXOR operator is expensive, the gates based on EXOR are the cheapest in quantum technologies, because of the similarity of this gate to the interaction of particles (see section 2.2 in chapter 2). Note also that the gates that use OR are expensive and unnecessarily large in quantum implementation, because they are ultimately realized based on the Boolean logic law $\mathbf{a + b = a \oplus b \oplus ab}$.

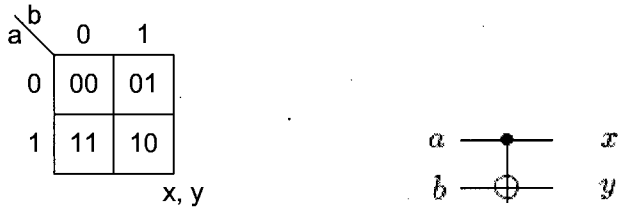
3.1.3. The AND/EXOR base of logic. Fundamental methods and graphic visualizations.

3.1.3.1. Quantum Karnaugh Maps.

Now we will bring the point of importance and difference of AND/EXOR base of logic. When analyzing such circuits it is important to use the familiar Karnaugh maps (K-maps) in a new way. The user has to learn how to overlap groups in the map – this way new circuits and new circuit types have been invented in our PSU group. We will use many KMaps in this thesis; standard KMaps, and their generalizations presented later on. These maps allow to find patterns in Boolean, multiple-valued, multiple-valued-input-binary-output and quantum functions. All synthesis methods in classical logic are based on patterns, the special classes of functions (such as the symmetrical or unate functions) have their specific patterns in KMaps. Therefore, being able to find new

types of patterns and use these patterns in synthesis is very important when one wants to create new logic synthesis methods for new types of logic.

The Karnaugh map is derived from the truth table in a relatively simple process. The Karnaugh map of the CNOT gate is illustrated in Figure 3.1.3.1.1.



(a) (b)

Figure 3.1.3.1.1: a) Complete Karnaugh map of the CNOT Gate from Figure 3.1.3.1.1b

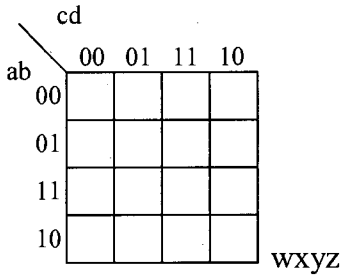


Figure 3.1.3.1.2: Skeleton of the 4 bit Karnaugh maps.

The arrangement of bits on the Kmap's rows and columns are in a sequence known as Gray code, where each value is only one bit change away from the preceding value. In this case, the procession is 0,1. The sequence is 00,01,11,10, as it is for all two-bit

Karnaugh maps (an example is in Figure 3.1.3.1.2), and so on. In a Karnaugh map, each possible bit combination of a and b is listed, with cells representing every single possible input/output combination. Use the truth table to put the correct output in each cell. We will notice that the Karnaugh map for 2 inputs registers x and y as the outputs (Figure 3.1.3.1.1a). Now we make it y Karnaugh map (Figure 3.1.3.1.3) and synthesize from it (other output is trivial).

For EXOR-based synthesis, groups in the map are "boxes" (loops) that should include as many ones as possible in it, and can overlap. Assume that zero is an even number. Thus every zero of the Kmap should be covered by an even number of groups (as using these rules: $A \oplus A = 0, A \oplus 0 = A$). Every one of the KMap cells with a "1" should be covered by an odd number of groups. The AND/EXOR synthesis methods differ only in the strategy how these groups are selected. In this case, we can have one-cell groups; in all larger Karnaugh maps, the groups must have a power of two of cells so you can write their logic expressions. The logic expression (logic code) of a group is based on the nature of the cells it occupies. It represents a product of literals (inputs and negated inputs).

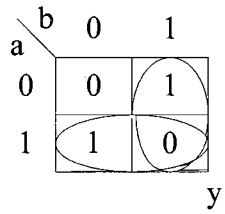


Figure 3.1.3.1.3: Groups in partial Karnaugh map of CNOT. Overlap of the groups represents 0. Thus function is $\bar{a}b \oplus a\bar{b} = a \oplus b$.

During synthesis, we can take the notations for each of the groups and EXOR them all together, then try to simplify them algebraically. As the cells in the groups cover a and b, and they both overlap over a 0, the notation is $a \oplus b$, or an EXOR of a and b. (Figure 3.1.3.1.3) Through a Karnaugh map, we can derive the function of a gate whose behavior was specified by this KMap. This simple principle is the base of all new synthesis methods introduced in next chapters.

Thus we can see that the circuitry of a function can be found through the utilization of Karnaugh maps and logic synthesis, leading to a quantum circuit. For any desired function, we can write the Karnaugh map based on how the desired function transforms inputs into outputs. From there, the designer can use groups of KMap cells and logic synthesis procedures to simplify the function into a collection of basic functions (OR, AND, EXOR) and so the designer can derive the circuitry of the desired function specification. Although KMap is useful to invent new methods and was used by me extensively in this dissertation, it is only a means to design an efficient computer algorithm that executes the entire synthesis. Thus the role of a human is not to design quantum circuits using KMaps but to develop software to design quantum circuits and the role of (quantum) KMaps is of a didactic nature only.

The Kmaps are useful in designing classical and reversible circuits, although in reversible and quantum logic other authors do not use them. As we illustrate in this thesis, we found a way to use Kmaps also in truly quantum (non-permutative) circuit synthesis. We call them Quantum QMaps and they were introduced for the first time in this dissertation.

3.1.3.2. From reversible gates to quantum gates.

3.1.3.2.1. Superposition and its visualization in Kmap.

In quantum computers, one is allowed to use only quantum states instead of the classical states. So, the electric spin or polarization can be replaced by some quantum state: the quantum bit (qubit for short). Just as a bit has a state 0 or 1, a qubit also has a state $|0\rangle$ or $|1\rangle$. The difference between bits and qubits is that a qubit $|\gamma\rangle$ can also be in a linear combination of states $|0\rangle$ and $|1\rangle$:

$$|\gamma\rangle = \alpha|0\rangle + \beta|1\rangle$$

This above equation is in the so-called Dirac notation which is the standard notation for states in Quantum Mechanics.

The state $|\gamma\rangle$ above is called a superposition of the states $|0\rangle$ and $|1\rangle$ with amplitudes α and β (α and β are complex numbers). Thus, the state $|\gamma\rangle$ is a vector in a two-dimensional complex vector space with basis vectors $|0\rangle$ and $|1\rangle$. The matrix (Heisenberg notation) representations of the vectors $|0\rangle$ and $|1\rangle$ are given by

$\begin{bmatrix}1\\0\end{bmatrix}$ for State $|0\rangle$, $\begin{bmatrix}0\\1\end{bmatrix}$ for State $|1\rangle$. Thus $|\gamma\rangle = \begin{bmatrix}\alpha\\0\end{bmatrix} + \begin{bmatrix}0\\\beta\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$ is a vector of complex amplidudes.

Quantum mechanics tells us that if one measures the state $|\gamma>$ one gets either $|0\rangle$, with probability $\alpha\alpha^*$ ($|\alpha|^2$), or $|1\rangle$ with probability $\beta\beta^*$ ($|\beta|^2$). Here, α^* is the complex conjugate of α. If α was a complex number g + bj, the conjugate would be g – bj ($j^2 = -1$). That is, measurement changes the state of a qubit. In fact, any attempt to find out the amplitudes of the state $|\gamma>$ produces a nondeterministic collapse of the superposition to either $|0\rangle$ or $|1\rangle$ basis states (eigenvectors). If $|\alpha|^2$ and $|\beta|^2$ are probabilities and there are only two possible outputs, then the calculation as in Figure 3.1.3.2.1.1 can be done.

Sum of all event's probabilities is "1" so that

$$|\alpha|^2 + |\beta|^2 = 1$$

$$\begin{cases} \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle & \textit{Supperposed Sate} \\ \alpha|0\rangle + \beta|1\rangle \end{cases}$$

$$\begin{cases} |\alpha|^2 = \alpha \bullet \alpha^* \rightarrow |0\rangle & \textit{Basic State} \\ |\beta|^2 = \beta \bullet \beta^* \rightarrow |1\rangle \end{cases}$$

Figure 3.1.3.2.1.1: Explanation of superposed states and their measurements.

3.1.3.2.2. Calculating a quantum state using matrices.

Any quantum circuit, however large, can be represented as a unitary matrix, which is multiplied by the input vector to generate the output vector, shown in Heisenberg and next in Dirac notations in Figure 3.1.3.2.2.

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Figure 3.1.3.2.2: Matrix representation of state 0 going through Hadamard gate. The Dirac notation is presented at the right.

In Figure 3.1.3.2.2 one can see how an input state reacts to the gate represented in matrix form. What is shown is the input vector, state 0, is acted upon by the Hadamard gate. When a circuit (Operator, Matrix) acts upon an input vector, it is simply multiplied by the matrix of the circuit, following the rules of standard matrix multiplication. The Dirac notation at the right is more convenient for some symbolic calculations and interpretation. We will be therefore using both Heisenberg and Dirac notations in this dissertation.

3.1.3.2.2.1. Calculating the operation matrix.

Given the means of calculating a gate's matrix as given above, to find the operation matrix is not too difficult. The most essential part of this is how to deal with parallel gates. In a circuit, gates will be found "on top" of each other, in terms of wiring (levels of qubits). As we remember from section 2.2, these gates are to be Kronecker Product multiplied from top to bottom. Kronecker multiplication of two gates entails the second matrix being multiplied by each element in the first, with the solution replacing the element of the first. In Figure 3.1.3.2.2.1.1 we illustrate Kronecker type of multiplication on binary matrices. Observe that these matrices can be of arbitrary dimensions, allowing thus to mix binary and ternary qubits into a single unitary matrix.

$$A \otimes B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Figure 3.1.3.2.2.1.1: Example of Kronecker multiplication of 2×2 matrix A and 3×3 matrix B. This corresponds to a binary qubit on top and a ternary qudit (qutrit) on the bottom.

Kronecker Products will create a large matrix for the first set of parallel gates of the circuit. Use this method until every set of parallels has its own matrix, and then multiply the matrices by each other, starting from the rightmost column towards the leftmost. Once this has been done, the operation matrix of the entire circuit will have been found. We use Matlab to perform all calculations on matrices larger than 8*8.

Many circuits results from this dissertation using Matlab [MATLAB] or QuiddPro software [QuIDDPro]. Some quantum algorithms were also verified.

3.1.3.3. States calculated by the Hadamard gate.

As we remember, the Hadamard gate is represented by a 2-by-2 matrix from Figure 3.1.3.3.1. Applying the gate to states $|0\rangle$ and $|1\rangle$ we obtain states that in Dirac notation are shown in Figure 3.1.3.3.2. How we can draw the superposed states created by this gate in a quantum Kmap?

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figure 3.1.3.3.1: The Hadamard gate matrix.

$$\begin{aligned} H|0\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \\ H|1\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \end{aligned}$$

Figure 3.1.3.3.2: Dirac notation of Hadamard outputs.

The Hadamard gate followed directly be the measurement gate acts like an ideal random number generator, with one input. When the Hadamard gate operates on inputs $|1\rangle$ or $|0\rangle$, the resulting outputs after measurement will be identical. Though the result for $|1\rangle$ has a $-|1\rangle$ entry instead of $|1\rangle$, this is irrelevant in measurement since all probability amplitudes are squared if the output of H is directly measured (i.e., the

global quantum phase is lost). The output state before the measurement (see Figure 3.1.3.3.2) represents an equal probability of states $|1\rangle$ and $|0\rangle$, but it represents also the phase. As the coefficient becomes the amplitude of both states, the square of it (1/2) becomes the probability of that state in case of measurement. In this case the phase is not relevant. However, before the measurement some next operations can be executed on this state so its phase is relevant in such a case, this is so for instance in Grover algorithm. Therefore the KMap of the Hadamard gate, shown in Figure 3.1.3.3.5, illustrates complete information about the output quantum states for all possible input basis states. Observe that this quantum KMap is just another form of illustrating a quantum state which can be done by all output quantum vectors. KMap has however more information than the truth tables or vectors. This information is useful in analysis and synthesis processes to those users who understand well functional patterns in classical KMaps. Let us observe that the entries in the binary cells of the KMap are no longer binary but may be superposed or even entangled values.

In Figure 3.1.3.3.3 a Superposition state created by the Hadamard gate is shown. Figure 3.1.3.3.4 repeats these calculations using the Heisenberg notation. As often done by physicists, the coefficient $\frac{1}{\sqrt{2}}$ is omitted in this particular calculation.

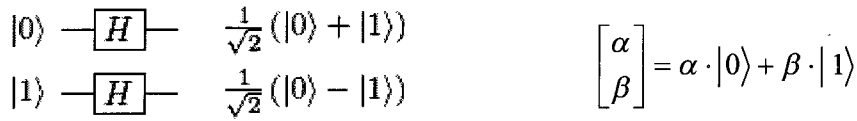


Figure 3.1.3.3.3: The symbolic notation for a Hadamard gate that is controlled by various basis states.

$$\textit{Hadamard apply to } |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\therefore \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = |0\rangle + |1\rangle$$

$$\textit{Hadamard apply to } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = |0\rangle - |1\rangle$$

Figure 3.1.3.3.4: Analysis of Hadamard gate applied to various input states.

Figure 3.1.3.3.5. illustrates the quantum K-map of the Hadamard gate.

0	$0.7071\|0\rangle + 0.7071\|1\rangle$
1	$0.7071\|0\rangle - 0.7071\|1\rangle$

Figure 3.1.3.3.5: The Quantum Kmap of the output of Hadamard gate (from Matlab).

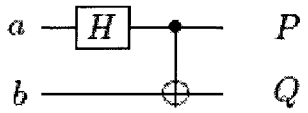


Figure 3.1.3.3.6: The EPR circuit that illustrates the concept of entanglement.

The famous EPR circuit illustrating the thought experiment of Einstein, Podolsky and Rosen is given in Figure 3.1.3.3.6. and its corresponding quantum K-map in Figure 3.1.3.3.7. This table has been verified using Matlab as in Figure 3.1.3.3.8.

a \ b	0	1
0	$\frac{1}{\sqrt{2}}\|00\rangle+\frac{1}{\sqrt{2}}\|11\rangle$	$\frac{1}{\sqrt{2}}\|01\rangle+\frac{1}{\sqrt{2}}\|10\rangle$
1	$\frac{1}{\sqrt{2}}\|01\rangle-\frac{1}{\sqrt{2}}\|10\rangle$	$\frac{1}{\sqrt{2}}\|00\rangle+\frac{1}{\sqrt{2}}\|11\rangle$

P,Q

Figure 3.1.3.3.7: The quantum KMap illustrating the output state of the EPR circuit. This KMap visualizes the entanglement from the circuit in Figure 3.1.3.3.6.

a	b = 0	b = 1
0	0.7071 $\|00\rangle$	0 $\|00\rangle$
	0 $\|01\rangle$	0.7071 $\|01\rangle$
	0 $\|10\rangle$	0.7071 $\|10\rangle$
	0.7071 $\|11\rangle$	0 $\|11\rangle$
1	0.7071 $\|00\rangle$	0 $\|00\rangle$
	0 $\|01\rangle$	0.7071 $\|01\rangle$
	0 $\|10\rangle$	- 0.7071 $\|10\rangle$
	- 0.7071 $\|11\rangle$	0 $\|11\rangle$

Figure 3.1.3.3.8: Matlab simulation to find the Quantum KMap for EPR circuit.

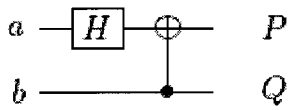


Figure 3.1.3.3.9: A circuit similar to EPR circuit but the "EXOR down CNOT" was replaced with the "EXOR Up CNOT".

The circuit from Figure 3.1.3.3.9 has been also verified with Matlab (Figure 3.1.3.3.10).

As we see, by rotating the CNOT gate the entanglement is removed, as the quantum states from Figure 3.1.3.3.10. can be factorized to separate qubit states.

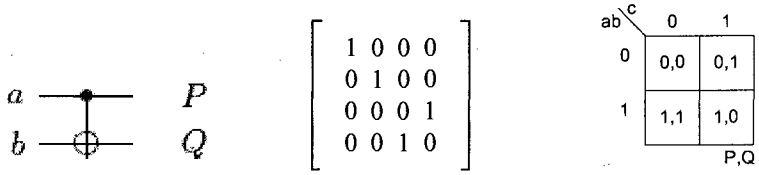
a \ b	0	1
0	0.5000 – 0.5000i 0 0.5000 – 0.5000i 0	0 0.5000 – 0.5000i 0 0.5000 – 0.5000i
1	0.5000 – 0.5000i 0 -0.5000 +0.5000i 0	0 -0.5000 +0.5000i 0 -0.5000 – 0.5000i

Figure 3.1.3.3.10: Matlab simulation QMap for the circuit when CNOT is controlled from the bottom bit(Figure 3.1.3.3.9). There is no entanglement.

3.1.4. Visualization of states in larger gates.

3.1.4.1. The Feynman or CNOT gate

For illustration we will compare various notations for the same gate. This is the CNOT gate from Figure 3.1.3.3.6 used in EPR circuit above. Its permutative matrix is 4-by-4, as shown in Figure 3.1.4.2.2.1a and its KMap is shown in Figure 2.4.4.2.2.1b. Please compare the matrix and the KMap. Remember that the order in rows and columns in the matrix is natural binary code and not the Gray code as in KMaps.



a) *b)* *c)*

Figure 3.1.4.1.1: (a) Feynman gate, (b) Feynman gate matrix, (c) the KMap of the Feynman gate.

Many of CNOT properties have already been seen above, in the Quantum Circuitry section. It is basically a reversible EXOR gate, reversible in that each qubit is continued to an output, unlike the classical EXOR. It is also deterministic, unlike the Hadamard, which means that a given input vector will always register the same output value. This gate is inexpensive in quantum and thus making it the base of synthesis is one of the main ideas of this thesis.

3.1.4.2. The 3*3 Toffoli or CCNOT gate

The Toffoli is an interesting and powerful gate in that it can have any number of inputs and the EXOR can be located in any wire of it. To be of practical usage, it must take these many forms. The circuitry is as in Figure 3.1.4.2.1:

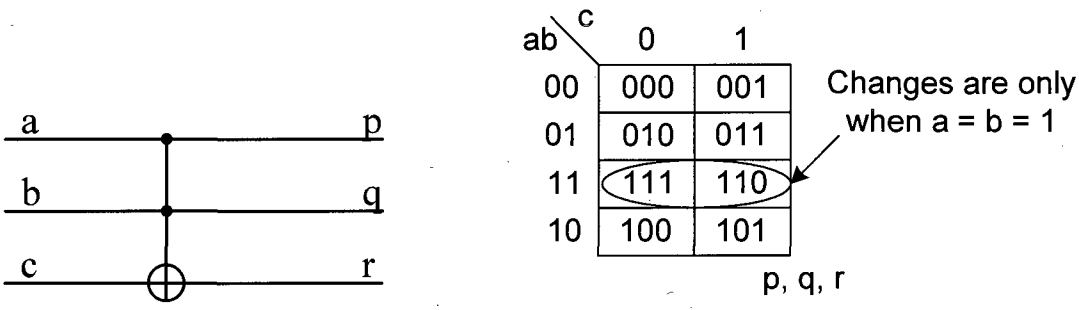


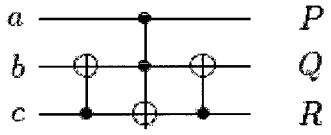
*Figure 3.1.4.2.1: The 3*3 Toffoli gate. It is also called the Controlled-Controlled-NOT or the CCNOT gate. The right part of the figure shows the Kmap for this gate.*

We can see that it is a double controlled inverter. One might think that the addition of another control would still make it a close relative of the Feynman. That is not so. For the Toffoli has 3 inputs, a, b, and c, and the designer can put constants in any of those positions, thus transforming the gate. By manipulations of this property, one can derive classical gates, and thus, prove that the Toffoli is a universal quantum gate.

The input/output relationship is p = a, q = b and r = ab$\oplus$c. Although Toffoli is a generalized form of the Feynman gate, the Toffoli gate is a universal gate in both classical and reversible (but not quantum) logic but the Feynman gate is not universal.

On the other hand Feynman gate is Affine gate but Toffoli gate is not. These gates are then complementary and using them together leads to a synergy.

3.1.4.3. The 3 * 3 Fredkin or Controlled-SWAP gate



$P = a,$
$Q = (b \oplus c) \oplus (ab \oplus c\bar{a}) = b\bar{a} \oplus ca$
$R = a.(b \oplus c) \oplus c = ab \oplus c\bar{a}$

Figure 3.1.4.3.1: Fredkin gate realized using Toffoli and CNOT gates. At right we illustrate algebraic analysis method using Boolean and EXOR algebra.

Fredkin gate in quantum array form is analyzed as in Figure 3.1.4.3.1. It is important to note that one can view the Fredkin gate from a different perspective, other than AND/EXOR logic. This perspective is that of multiplexing between signals. This perspective on the Fredkin gate (not AND/EXOR logic) is used in Cellular Automata and Optical realizations and in some nano-technologies especially those based on billiard ball model and conservative logic. This point of view is illustrated in Figure 3.1.4.3.2.

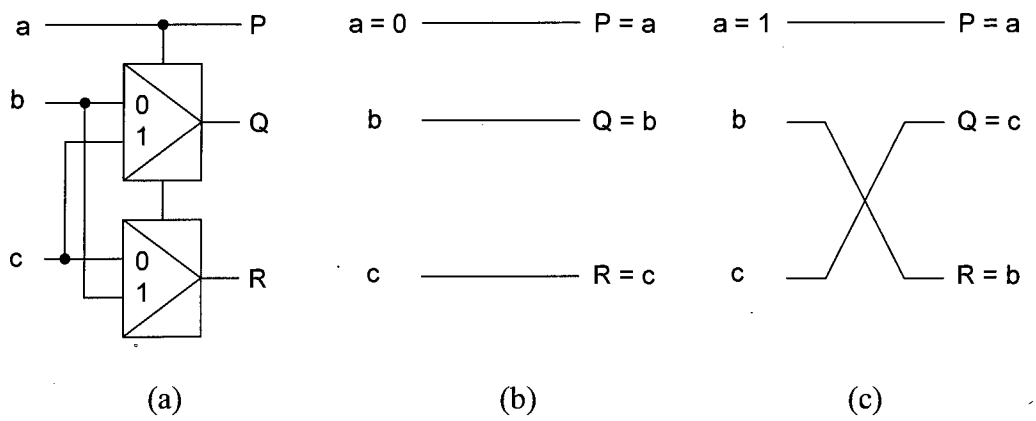


Figure 3.1.4.3.2: (a) Fredkin gate represented symbolically with classical Multiplexers, (b) Fredkin gate at control input value a = 0, (c) Fredkin gate at control inputvalue a = 1.

Understanding the Fredkin from multiplexers we can generalize Fredkin to arbitrary number of qubits. See Figure 3.1.4.3.3. below. Considerations like this have been used by us to create new synthesis methods.

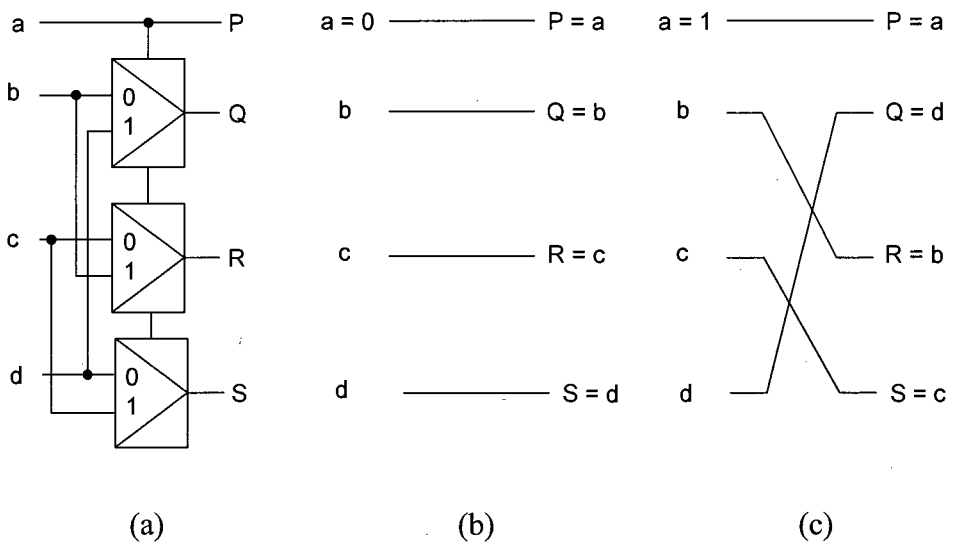
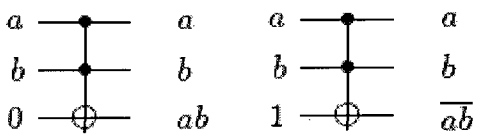


Figure 3.1.4.3.3:(a) Generalized Fredkin Gate using classical multiplexers. (b) What Generalized Fredkin gate realizes while control input a = 0 and (c) What Generalized

Fredkin gate realizes when control input a = 1.

3.1.4.4. The Ancilla Qubits

Ancilla qubits are extra qubits. They are not variables, though they can be mapped onto an output. Ancilla qubits are useful for input variables in large size gates, as well as on wires that lead to the output. In a large circuit, it is not always good to have every wire assigned to a variable input; the functions of the gates can be changed in useful ways if some of the wires are assigned to a constant. To explain its uses in large gates, one must look no further than the Toffoli. In order for the Toffoli to be of use, in many cases the wire that goes to the EXOR must have a constant value (1 or 0) to change its uses and allow it to be a universal gate. Those 1's and 0's are ancilla bits, since they are not input variables, and are constant. They can also be placed on wires leading to an output, whether it is because the ancilla bit was on the answer register of the final gate, or because it is simply more efficient to do so. Figure 3.1.4.4.1 illustrates how AND and NAND gates of classical logic can be built using the Toffoli gate with the lowest qubit being an ancilla bit. As we see in the example, ancilla bit is absolutely necessary if I want to convert a non-reversible function (called also an irreversible function) like AND or EXOR into reversible (quantum) circuit.



(a) AND (b) NAND

Figure 3.1.4.4.1: (a) Realization of AND gate using Toffoli gate with the ancilla qubit initialized to zero, (b) Realization of NAND gate using Toffoli gate with the ancilla qubit initialized to one.

3.2. Why the AND/EXOR Logic Base?

3.2.1. Is the AND/EXOR base best for reversible and quantum logic?

While not as widely utilized for classical integrated circuit design as the AND-OR Sum-of-Product (SOP) logic, the Exclusive-Or Sum-of-Product (ESOP) form (the most general, unrestricted AND-EXOR logic form) compares favorably even in classical design [Sasao90c, Sasao91d, Sasao91e]. Functions realized by such circuits (ESOPs) can have fewer gates, fewer connections, and take up less area even in the VLSI and especially, FPGA realizations. More importantly, in case of quantum arrays, the advantage of ESOP over SOP becomes dramatic, as will be illustrated in the next chapter. (As an illustration one can take function f = abc + cde + gfe + klm and next convert it to ESOP. Here + stands for inclusive Boolean OR). AND-EXOR circuits are also easily testable [Reddy72, Sasao95g, Kalay99, Kalay99a]. It was shown, both theoretically and experimentally [Sarabi93, Sasao91c, Sasao91d, Sasao91e] that

ESOPs have on average smaller numbers of product terms for both "worst case" and "average" Boolean functions. Additionally, it can be shown that reversible circuits based on two-level AND-EXOR realizations are also good for the combinational logic portions of finite state machines, as they have proven more testable and can yield less area than the two-level AND-OR implementations. The same is true for quantum state machines assuming that the classical memory is used in them and quantum circuit is used only to calculate the next state and the output state. (Measurement units are inserted on all outputs of this circuit Figure 3.2.1.1). Thus it can be concluded that the AND-EXOR implementation is in many applications superior to the AND-OR logic, for both its testable and economical characteristics, and in quantum logic this type of logic remains practically the only logic of choice to design permutative circuits [Perkowski03].

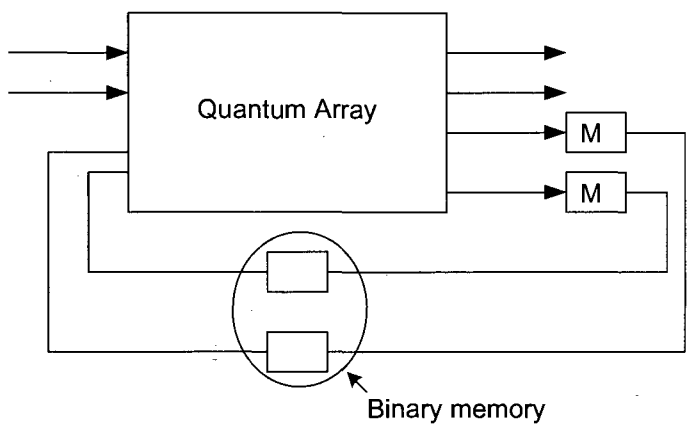


Figure 3.2.1.1: Realization of a Mealy Quantum State Machine with classical Binary memory. The Binary memory uses standard memory elements (flip-flops). The primary inputs and primary outputs are quantum. This design is based on a quantum array that may be designed and specified as in this and next chapters.

However, let us observe that other authors use other types of logic for reversible and quantum synthesis. Many authors including Igor Markov, Vivek Shende, Alexis De Vos, Yvan von Retergem, Guowu Yang, William Hung, Xiaoyu Song and Marek Perkowski use group theory which does not distinguish between AND/OR and AND/EXOR base. This is true and this is other possible line of research. But let me make a point that the group theoretical approaches are used so far only for small circuits, at most 4*4, while our methods are applicable to circuits with at least 14 bits. Some other authors such as Dmitri Maslov, Michael Miller and Gerhard Dueck use Fredkin gates but these gates are presented in the framework of AND/EXOR type multi-input CCNOT gates. Concluding, from the point of existing theories of representation and their corresponding algorithms there are two groups of algorithms used with some (limited) success – the group theory-based and the AND/EXOR-based and this thesis follows the more common AND/EXOR approach.

Let us observe, based on literature, that the only competitor universal gates to the Toffoli gates are the Peres and Fredkin gates. The Peres Gate has many EXORs inside it in every known realization as it can be composed from Toffoli and CNOT or from direct $CV/CV^{\dagger}$ realization shown previously, therefore this gate can be best handled with the methods developed in this dissertation. Fredkin's Gate internal realization in many quantum technologies is also based on the 2-Toffoli gate ($P=a$, $Q=b$, $R=ab \oplus c$) and two Feynman gates, so it is reducible to our methods.

Although we can handle Fredkin gates in terms of AND/EXOR logic, as in new variants of MMD [Miller03], it may be not the best way if one can realize this gate directly with electromagnetic pulses and the cost of such a realization would be smaller than its counterpart cost shown earlier. There are at least two interesting aspects related to Fredkin gates realization and cost:

1. In some technologies such as superconducting, the Fredkin gate is built inexpensively from Square root of Swap gates [Blais00]. This shows that not always AND/EXOR logic and ESOP-like structures are the best basic logic types and structures and EXOR may not necessarily be the best basic gate in quantum. We write about this fact just to show the wide scope of our literature search, but we are not addressing this issue much as it seems to require a totally new mathematical approach. The Fredkin gate design issues are discussed in a PhD of Nouraddin Alhagi [Alhagi08].

2. When realizing satisfiability formulas in form of a product of sums, there is no advantage or no possibility of converting them to ESOP, in this case the POS (product of sums) logic which is dual to SOP is still applicable, even as the price of many ancilla bits is paid.)

	Classical	Reversible	Quantum
NOT	@	@	@
AND	@		
OR	@		
EXOR	@	@	@
C/CNOT		@	@
CV			@
$CV^{\dagger}$			@
Hadamard			@

Table 3.2.1.1: Tabular Comparison of Classical, reversible and Quantum gates.

Finally, Table 3.2.1.1 compares classical, reversible and quantum gates. As we see, NOT gate is used in all technologies and is very cheap. It should be then used as much as possible in reversible and quantum synthesis, this leads to concepts of polarity and mixed polarity forms and expressions introduced in chapters 7 - 9. Next, the EXOR operator as such is cheap as a component of gates in all these technologies but especially in quantum. It should be used extensively in synthesis methods, which is not satisfied by other authors. CNOT gate and CCNOT gate are used in reversible and quantum but they are more expensive. The methods should thus allow to realize circuit with affine (EXORs and NOTs) gates as much as possible, and CCNOT only when absolutely necessary. CCNOTs are build from CV and $CV^{\dagger}$ gates and the Hadamard gate is the only one more quantum gate that we need.

3.2.2. Some types of Permutative Quantum Circuits. The Quantum circuit Synthesis problem

3.2.2.1. Forms for AND – EXOR Logic.

We can not build quantum circuits based on AND/OR gates without ancillas, as they are not reversible [Toffoli80]. If we convert such circuits (netlists from AND and OR and similar gates) directly to reversible logic – many ancilla bits must be in most cases added. This should be in general avoided. Researchers are emphasizing increasing efforts to find an automatic way to create efficient quantum circuits implementing Boolean functions [Lee99, Iwama02, Younes03]. We know however that there is a close connection between Boolean Quantum operations and certain classical Boolean operations known as Reed-Muller logic expansions [Almaini89]. The AND-EXOR form has been developed into a complete hierarchy of Reed-Muller (RM) expansions, using the Shannon, Positive Davio, and Negative Davio Expansions in the works of Tsutomu Sasao [Sasao91c, Sasao91d, Sasao91e, Saso93e, Sasao95g] and especially the PSU group (Perkowski, Mishchenko, Dill, Sarabi, Schaefer, Safranek, Pierzchala, Chrzanowska-Jeske) [Perkowski91, Dill97b, Sarabi92]. This hierarchy is described with logic equations, forms, trees, and decision diagrams [Sasao93e]. We will present this hierarchy for completeness of this dissertation and next we will add new items to the hierarchy, motivated by their practical applications in quantum NMR technology. In quantum interpretation the whole new extended hierarchy gets new meaning as a hierarchy of quantum array structure types that can be relatively easily mapped to the

recently proposed Quantum Field Programmable Gate Arrays [Nielsen97] and other quantum realization technologies. Our interest is mainly in the dominating technology of NMR but also to the close to it Ion Trap technology which is predicted to have a great future although it is less developed as of year 2008. As components of our oracles we are particularly interested in (multi-output) Fixed Polarity Reed Muller (FPRM), Generalized Reed-Muller (GRM) forms and their affine generalizations, because of their relative simplicity and usefulness in design of quantum circuits, complete oracles and quantum evolvable hardware. Although some of the forms from hierarchy have been the focus of the logic synthesis and minimization research for many years and investigated by many authors, finding the exact solutions for small circuits or the good quality approximate circuits is still very difficult for larger functions that are necessary for some practical oracles that use interative arrays of simple blocks (Chapters 11 and 15).

The GRM logic is a canonical expression (exhibiting a regular structure) which is a subset of the Exclusive-Or Sum-Of-Products (ESOP) expression, in which for every subset of input variables there exists at most one term with any polarities of variables. Explaining GRM, we should explain that implementing Boolean functions on quantum computers is an essential goal for us to explore the benefits that may be gained from systems operating by quantum rules. On classical computers, a circuit can be build for any Boolean function using *AND, OR and NOT* gates.

3.2.2.2. The Fixed-Polarity Reed-Muller Forms.

Any Boolean function f with n variables, f: $\{0, 1\}^n \rightarrow \{0, 1\}$, can be represented as a disjoint sum of products SOP [Almaini89] as in equation 3.2.2.2.1:

$$f(x_0,.....,x_{n-1}) = +\sum_{i=0}^{2^n-1} a_i m_i \qquad \textit{Equation 3.2.2.2.1}$$

Where m_i are the minterms and a_i = 0 or 1 indicates the presence or absence of minterms respectively and the plus in front of the sigma means that the arguments are subject to Boolean operation inclusive-*OR*. This expansion can also be expressed in Reed-Muller (RM) as in Equation 3.2.2.2.2 from [Akers59]:

$$f(\hat{x}_0,.....,\hat{x}_{n-1}) = \oplus\sum_{i=0}^{2^n-1} b_i \varphi_i \qquad \textit{Equation 3.2.2.2.2}$$

where

$$\varphi_i = \prod_{k=0}^{n-1} \hat{x}_k^{i_k} \qquad \textit{Equation 3.2.2.2.3}$$

where $\hat{x}_k = x_k$ and $x_k, b_i \in \{0,1\}$ and i_k represent the binary digit of k.

φ_i are known as product terms and b_i determine whether a product term is presented or not. Symbol $\oplus$ indicates the EXOR operation and multiplication is assumed to be the *AND* operation.

Consider the RM expansion shown in Equation 3.2.2.2.2, where $\hat{x}_k$ can be x_k or $\overline{x}_k$ exclusively. For *n*-variables expansions where each variable may be its true or

complemented form, but not both, then there will be 2^n possible expansions. These are known as the *fixed polarity Reed-Muller* (FPRM) *expansions*. We can differentiate various FPRM expansions by *polarity number*, which is a number that represents the binary number calculated in the following way: if a variable appears in its true form, it will be represented by 1, and by 0 for a variable appearing its complemented form. For example, consider the function $f(\hat{x}_0,\hat{x}_1,\hat{x}_2): abc \oplus a \oplus 1$ *where* $f(x_0,x_1,x_2)$ has polarity 7 (111), $f(x_0,\overline{x}_1,x_2)$ has the polarity 5 (101), $f(\overline{x}_0,x_1,\overline{x}_2)$ has polarity 2 (010), and $f(\overline{x}_0,\overline{x}_1,\overline{x}_2)$ has polarity 0 (000), and so on.

Younes and Miller [Younes03] showed that changing the polarity will change the number of *CNOT* gates in the circuit; and its efficiency.

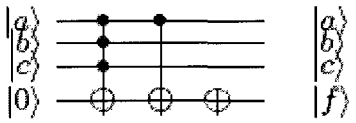


Figure 3.2.2.2.1: Quantum Circuit f for Polarity Number 7 for function f = abc ⊕ a ⊕ 1 .

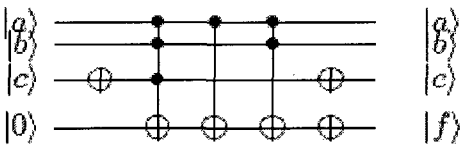


Figure 3.2.2.2.2: Quantum circuit f for Polarity Number 6 for function from Figure 3.2.2.2.1.

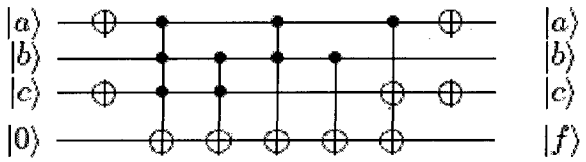


Figure 3.2.2.2.3: Quantum circuit f for Polarity Number 2.

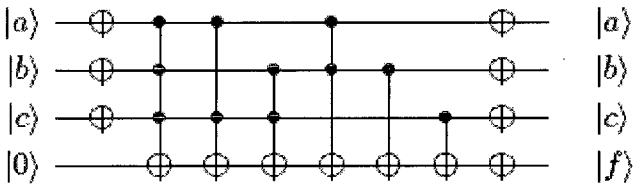


Figure 3.2.2.2.4: Quantum circuit f for Polarity Number 0.

For FPRM expansions, the number of *CNOT* gates in the final quantum circuit can be calculated as in Equation 3.2.2.2.4:

$S_1 = m + 2K, \qquad 0 \le m \le 2^n; \quad 0 \le K \le n$ *Equation 3.2.2.2.4*

where S_1 is the total number of CNOT gates, m is the number of product terms in the expansion, K is the number of variables in the complemented form and n is the number of inputs to the Boolean function; the term 2K represents the number of CNOT gates that will be added at the beginning and at the end of the circuit (complemented form) to negate the value of control qubit during the run of the circuit and to restore its original value, respectively.

3.2.2.3. Which forms and gates are best for quantum circuits?

Expansions and gates that are efficient for classical logic circuits are not necessarily so efficient for quantum circuits. Thus we find the research interest in our thesis to develop the search algorithms for optimizing FPRM, GRM and the newly invented affine expansions and corresponding expressions for quantum Boolean functions similar to those found for the classical digital circuit design.

In other words, each term in the GRM form (introduced formally in next chapter) is unique in both variable name and polarity. It is interesting to note that often the GRM forms may produce results with the number of terms very close to that of exact minimum ESOPs [Cohn62, Perkowski99b, Saul92, Wu96]. GRM forms are also even more easily testable than the general-purpose ESOPs [Sasao95g]. In case of the classical Binary logic, [Sasao95g] showed that the average number of products for GRMs is less than half of the respective PPRMs.

There are several speculations [Weiss01, Hollenberg04], however, that reversible logics similar to those presented here will become practical when the technological limits of sub-micron technologies are reached. Also, there are both technological reasons (for technologies such as Josephson Junction or resonant tunneling diodes) and mathematical reasons why some new logic operators or design structures may become preferable. However, this dissertation is constrained only to quantum

technology because this technology is the most mature, most interesting and most promising. *It is the quantum technology that proposes an entirely new prospect for computing and not only speeds-up the current computing model.* One of the reasons that we discussed mapping of permutative gates to cellular automata is that according to Professor Ed Fredkin, cellular automata may allow to create in the future a unified view of the world in which the same mathematics will be used for the quantum world and the macro-world of standard physics. (Although it was not shown by anybody how to map efficiently non-permutative circuits to reversible cellular automata, it still may be possible, but we are not concerned with this issue here). We believe thus that based on the previous research reviewed in this thesis we can formulate the statement – *"every universal model of permutative computing (binary and multiple-valued – described by any permutative unitary matrices) is realizable at the level of quantum phenomena".*

What may be nonsensically complex in contemporary CMOS-based circuits, may be the best choice in quantum technologies. The best example is the Hadamard transform. One-bit Hadamard transform requires only two Pauli Rotations internally so it is the cheapest "quantum gate" after the inverter in quantum design (inverter requires only one Pauli rotation) . In classical logic design the Hadamard transform used one multi-bit subtracter and one multi-bit adder being thus a big and complex block of many AND/OR level gates (see Chapter 11). Although for other quantum gates the

differences are not that dramatic as for the Hadamard gate, the problem is very characteristic when comparing quantum and classical circuit design: *"what is cheap in classical logic may be very expensive in quantum logic (like OR of many terms) and what is expensive in classical logic may be very inexpensive in quantum logic (like Hadamard)"*. This is an important observation that explains why the entire design with quantum gates should be deeply re-thinked and methods may be adapted from classical design only with an extreme care.

Concluding, we motivated above the AND-EXOR forms for quantum design based on their strong links to NMR gates, their high testability and the possibility of using mathematics to develop structures and algorithms. It is obvious that, like in standard CAD, our algorithms have to use some kind of search. But there are many methods to execute search, evolutionary algorithms or simulated annealing are just two well-known search approaches. We have therefore now to discuss in more detail the advantages and disadvantages of known search methods and relate them to circuit structures. Although choice of AND-EXOR logic seems obvious, the choices of its forms are less obvious. We will discuss them now.

3.2.3. The problem of good structure selection.

3.2.3.1. Polarized forms.

Let us continue our background material overview with the crucial observation: it is

not only important to optimize certain type of circuit, but we must be able to select a good circuit type (structure) for the given problem and the given technology. For instance the minimized ESOP oracle for function $f = abc + \overline{abc} = abc \oplus \overline{abc}$ is shown in Figure 3.2.3.1.1. It has only two product terms. Although it theoretically looks like an optimal solution as it has the exact minimum number of terms, its realization in Figure 3.2.3.1.3 with more realistic gate model shows that the quantum cost of this ESOP circuit is high. On the other hand the factorized GRM form of f (Figure 3.2.3.1.4) has 3 product terms but has a smaller quantum cost. The GRM circuit that is shown in Figure 3.2.3.1.5 is also cheaper than the ESOP circuit but the PPRM circuit (Figure 3.2.3.1.6) is even more expensive for any type of cost function.

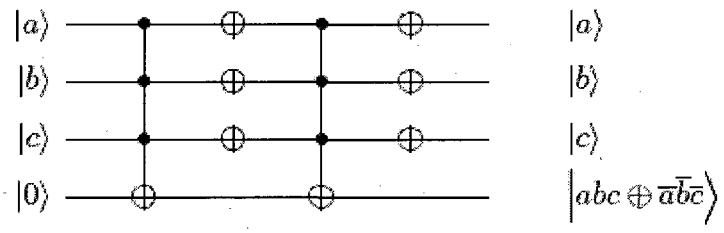


Figure 3.2.3.1.1: Quantum Oracle for function $abc \oplus \overline{a}\,\overline{b}\,\overline{c}$ *build as ESOP type expression realized with 4 * 4 Toffoli gates (non-existent technologically). These gates are decomposed to 2*2 controlled gates or 3*3 Toffoli macros which causes this solution to have a high quantum cost.*

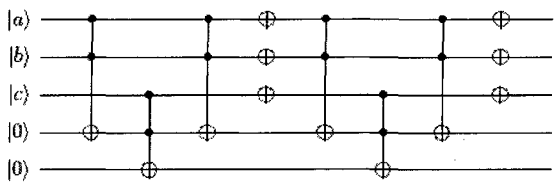
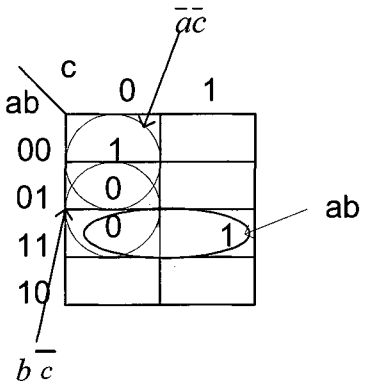


*Figure 3.2.3.1.2: Quantum Oracle for function from Figure 3.2.3.1.1 using realistic 3 * 3 Toffoli gates and one additional ancilla bit for the ESOP circuit from Figure 3.2.3.1.1.*

Each 3 * 3 Toffoli gate from Figure 3.2.3.1.2 can be realized as in Figure 3.2.3.1.3. This type of design allows for more realistic cost function estimation, but it is still far from the optimum. Observe in Figure 3.2.3.1.2 the NOT gates added at the end to return the original values of input variables, the condition is necessary in oracles, but is not necessary in blocks used only as parts of oracles.



$$f = \bar{a}\,\bar{c} \oplus b\bar{c} \oplus ab$$

Figure 3.2.3.1.3: KMap for the GRM realization of the function realized as ESOP in Figure 3.2.3.1.1.

Here we get a nice example which is ESOP realizing the function $f = abc \oplus \bar{a}\,\bar{b}\,\bar{c}$ in Figure 3.2.3.1.1, both two terms in ESOP here is the subset of {a,b,c}, which is allowed for ESOP. But for GRM, every term should be a different subset of

variables. Hence: the GRM will be $f = \bar{a}\bar{c} \oplus b\bar{c} \oplus ab$ in Figure 3.2.3.1.3, which is using subsets {a,c},{b,c} and {a,b}. This is not an FPRM circuit. ESOP uses more quantum primitives, thus it is expensive. In FPRM each variable is positive or negative, not both. GRM is different. GRM is mixed. In GRM, for every subset of variables, we have only one term. If the same subset of variables appears more than once, then it is not a GRM, perhaps ESOP. In Figure 3.2.3.1.1 to Figure 3.2.3.1.5, we nicely show the difference between ESOP based Quantum circuits and GRM based Quantum circuits visually.

We will discuss now how the better solution is found. The GRM for the function is done by EXOR-ing the three overlapping groups from Figure 3.2.3.1.3. After factorization, this leads to the realization of GRM as a quantum cascade from Figure 3.2.3.1.4. Without factorization, the GRM will lead to the oracle from Figure 3.2.3.1.2. Finally, the PPRM is shown in Figure 3.2.3.1.6. Obviously the PPRM is very expensive not only using quantum cost but also counting the gate number. Even better solutions for this kind of problems will be showed in the chapter 7 where I will introduce one of the main concepts of this dissertation – the affine gates. Solutions with affine gates are always better than the classical AND/EXOR solutions, provided that the sufficient search was executed to find these affine solutions.

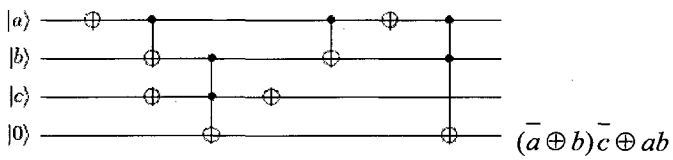


Figure 3.2.3.1.4: Realization of quantum cascade (oracle) for factorized GRM $f = \bar{a}\,\bar{c} \oplus b\bar{c} \oplus ab$ *(its KMap illustrated in Figure 3.2.3.1.3).*

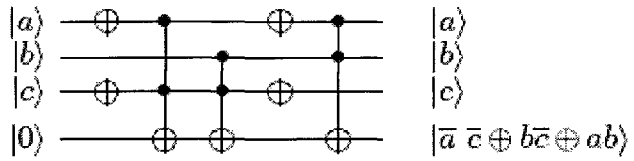


Figure 3.2.3.1.5: Quantum Oracle for direct (non-factorized) realization of GRM.

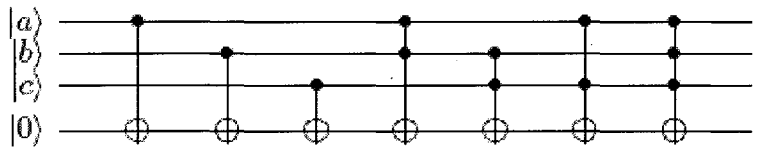


Figure 3.2.3.1.6 The quantum circuit (being also an oracle since inputs are replicated to output) for the PPRM form of function from Figure 3.2.3.1.1.

The PPRM circuit from Figure 3.2.3.1.6 is nonsensically non-optimum but demonstrates how important is a good selection of circuit model and polarity in practical quantum design. In case of a circuit with many inputs and outputs the quantum cost differences may be truly dramatic.

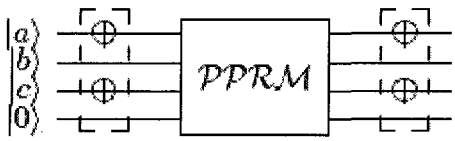


Figure 3.2.3.1.7: A general view of quantum oracle realizing an FPRM form. The circuit is a result of its polarity (NOT gates in front and in back) and its general gate/circuit type (PPRM realized with Toffoli gates in this and previous figures).

Finally, Figure 3.2.3.1.7 presents the general view of an FPRM circuit realized as a quantum array – it is a PPRM of some other polarity function surrounded by NOT gates. In this particular example the NOT gates are realized for qubits x_1 and x_3. This view is used in all synthesis algorithms introduced in this dissertation. The reader should appreciate from these examples, that changing the polarity influences very substantially the cost and especially the quantum cost of the solution. However, Figure 3.2.3.1.7 shows that the polarity is a global concept, the NOT gates affect the function inside the box PPRM in Figure 3.2.3.1.7. But these additional NOT gates cost very little, since in every quantum technology of implementation the cost of the NOT gate is practically negligible. Thus the circuit inside the box can be realized using any AND/EXOR method or affine gate based method to further decrease the entire realization cost.

As we see, every AND/EXOR synthesis method from this sub-area has thus two components:

(1) The polarity,

(2) The basic gate/circuit model of the circuit inside the box. In particular these can be Toffoli gates or affine gates of any type.

We are adding hereby the third component of "affine design" as the main innovative idea of this dissertation.

3.2.4. ESOP expressions

A question may arise: "why to use the concept of polarity at all?" May be removing this restriction one can create better circuits? Yes, in classical logic removing all polarity restrictions leads to the so-called ESOP or Exclusive-Or-Sum-Of-Products (non-canonical) circuits which have smaller number of terms. However, synthesis of such circuits, especially to minimize not only the number of terms but also the number of literals is extremely difficult. Also their testability is lower then that of the canonical forms. As we will see in future chapters, the ESOPs may be also worse for quantum realizations, especially for large functions with many don't cares. Thus in this dissertation we are not optimizing ESOP structures. In any case, one has to be familiar with them as we use them in few of our illustrative oracles in chapters 11 -15.

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

Figure 3.2.4.1: KMap with groups selected for ESOP expression for function F2. Overlap of even number of groups creates a "0".

Figure 3.2.4.1 shows KMap of realization of function F2 = $\overline{c}\,\overline{d} \oplus \overline{a}\,\overline{b} \oplus ac$ using ESOP expression. The principle of creating value zero in the overlap of groups is again illustrated. All next methods in this dissertation will use this principle. The quantum array for the formula above is shown in Figure 3.2.4.2. Please observe that many inverters are added, but they do not contribute much to the cost in any quantum realization technology known to me. F2 expression above is also a GRM, but this example better than the example from section 3.2.3.1 illustrates the synthesis approach and the repeated inverter characteristic to realization of ESOPs in quantum arrays.

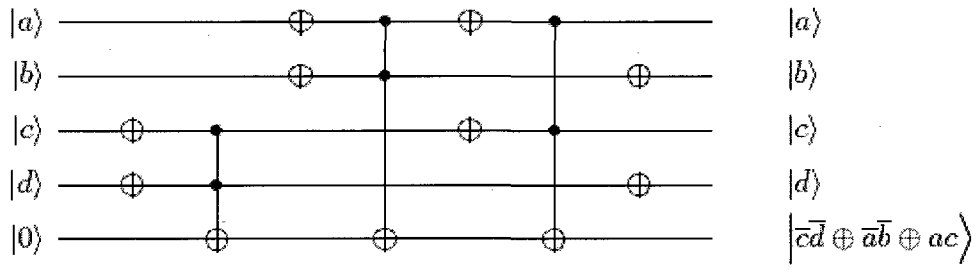


Figure 3.2.4.2: Quantum Array for function F2 from Figure 3.2.4.1 used as an oracle. This explains why two NOTs are added in qubits $|b\rangle$ and $|d\rangle$ - this is because we want to return original inputs at the output of every quantum oracle.

3.3. Motivating Example: Building a quantum array for a very simple oracle.

Now that we know how to realize permutative quantum circuits, we can show, ahead of order, a single example of building an oracle, just to show my thesis direction and explain many ideas of the thesis to which we refer in early chapters, before the oracles will be formally introduced in chapter 12.

The problem is this. We want to color nodes of the graph from Figure 3.3.1 below with as few colors as possible so that any two nodes linked by an edge have different colors. Assuming that we have no any knowledge of the graph that we color other than that it has five nodes, we have to assume pessimistically that in the worst case it needs as many colors as there are nodes, which means five. Five numbers need at least 3 bits to encode them, it would be too bad to have this kind of a problem for a graph with 10,000,000 nodes which would be colorable with 2 colors, but let us make important point again that we have absolutely no information about the data in this variant. However, if we would know that the graph is planar, one can use the famous "Four Color Theorem" to know that only four colors are sufficient and thus encode the colors with only two qubits.

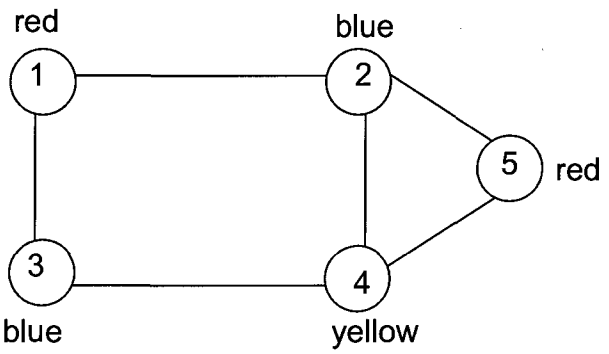


Figure 3.3.1: Graph for coloring with five nodes. It is colored with red, blue and yellow colors in such a way that every two neighbor nodes have different colors. The chromatic number of this graph is 3.

Assuming no knowledge of the chromatic number of the graph the encoding requires three bits for each color and is shown as in Table from Figure 3.3.2 below. One particular example of encoding another simpler graph is shown in Figure 3.3.5.

Color	**Bit**
red	a_1, a_2, a_3
blue	b_1, b_2, b_3
blue	c_1, c_2, c_3
yellow	d_1, d_2, d_3
red	e_1, e_2, e_3

Figure 3.3.2: Assignment of bits to encoded colors of nodes for the graph from Figure 3.3.1.

An inequality comparator circuit is used to compare two nodes of the graph, as shown in Figure 3.3.3 for nodes a and b. Such comparator is connected to encoding bits of any two nodes that are linked by an edge in the graph. If the colors of nodes a and b are the same then the output of the comparator will be zero. If the codes are different (which is good) then the output will be 1. Therefore, if oracle has such a comparator for every two nodes of the graph linked by an edge and if a global AND gate of outputs of comparators is created, the output of this AND gate will be one for a good coloring and will be a zero even only in one pair of neighbor nodes of the graph the proper coloring will be violated, see Figure 3.3.6 for the classical oracle.

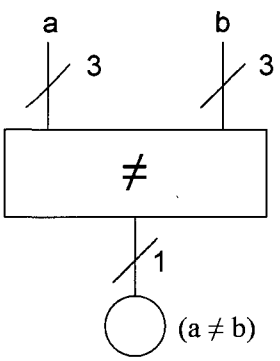
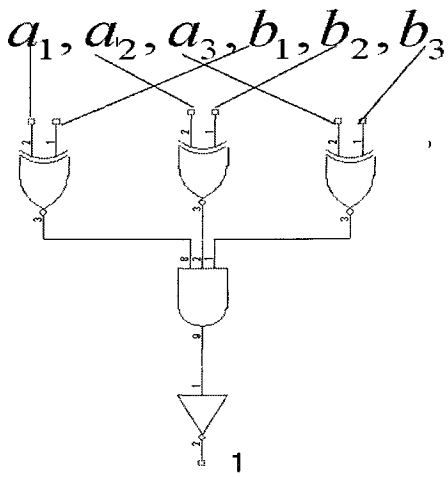
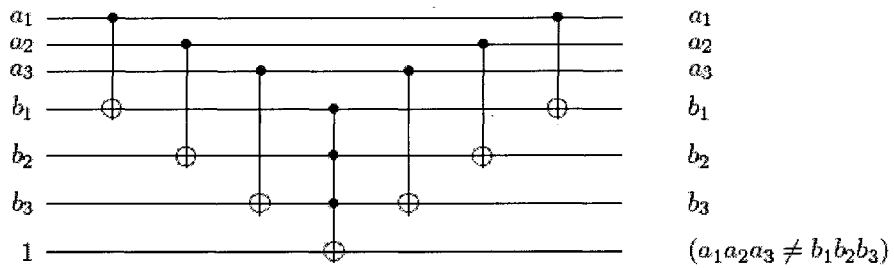


Figure 3.3.3: The inequality comparator used in Map Coloring and Graph Coloring problems. Here it compares node a with node b. Observe that the size of this comparator depends very much on the possible maximum number of colors. The comparator produces "1" at its output if the arguments a and b are different binary vectors of width 3. The binary signal (a ≠ b) is also called a predicate.



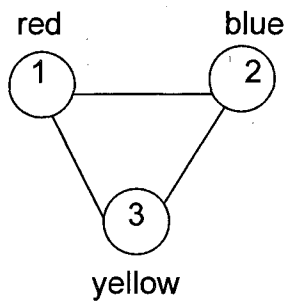
(a)



(b)

Figure 3.3.4: (a)The inequality comparator from Figure 3.3.3 applied assuming five or more (≤ 8) colors in the graph. This is a Classical schematic for the inequality operator circuit, but next we convert it to a quantum reversible circuit. (b) The quantum array for the comparator from Figure 3.3.4a. This is an oracle so three CNOT gates are added at right to restore inputs.

The classical schematics of the comparator using EXOR, NOT and AND gates is shown in Figure 3.3.4a. It is rewritten to the quantum array shown in Figure 3.3.4b.



red	000	n_1
blue	001	n_2
yellow	010	n_3

Figure 3.3.5: Encoding of colors for the graph coloring oracle of another graph having 3 nodes. This graph is used in Matlab simulation.

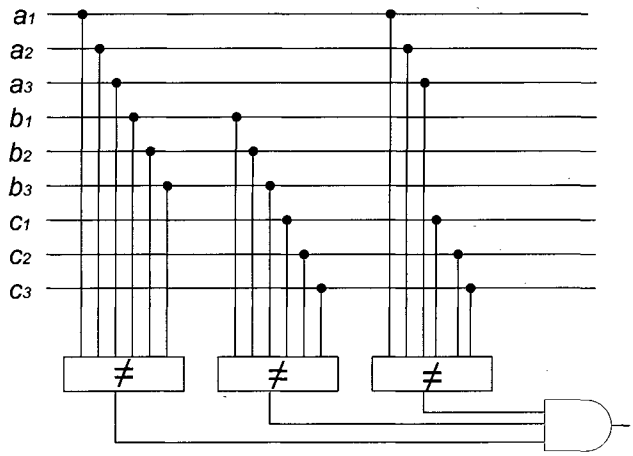


Figure 3.3.6: Principle of graph coloring applied to a simple graph from Figure 3.3.5. This is a classical oracle. In this and previous graph coloring problem we are not checking for a minimal solution. We look here only for a coloring that satisfies the constraint of correct coloring. Thus every proper coloring that uses any 3 of 5 colors is good. (this example is trivial, but we wanted to have a simple circuit for the example).

The final quantum oracle for Grover algorithm for the graph from Figure 3.3.5 is shown in Figure 3.3.7. It is preceded with Hadamard gates that create superposition of all input states corresponding to all possible colorings of the graph. The oracle is the part of the so-called Grover loop quantum circuit which includes some other output-

processing circuit and is repeated many times in the full Grover algorithm (Figure 3.3.8), which will be discussed in full detail in chapter 5. At this point our only goal was to explain the concept of a quantum oracle and its design using quantum gates. Remember that using reversible non-quantum gates is not possible in the oracle for Grover algorithm, because they would not produce and process the superpositions of quantum states which are fundamental to the Grover algorithm. In this example the oracle is very simple and can be designed by hand. In general, the oracle is very complex, its design will require automation and the thesis produces software (classical) and hardware/software (quantum) tools for this automation.

I believe that in future high level languages will be developed that will automatically design, adapt and reconfigure oracles thus the user will program in them without realizing the complexity of created circuits, as it is now in case of VHDL programming for ASIC or FPGAs.

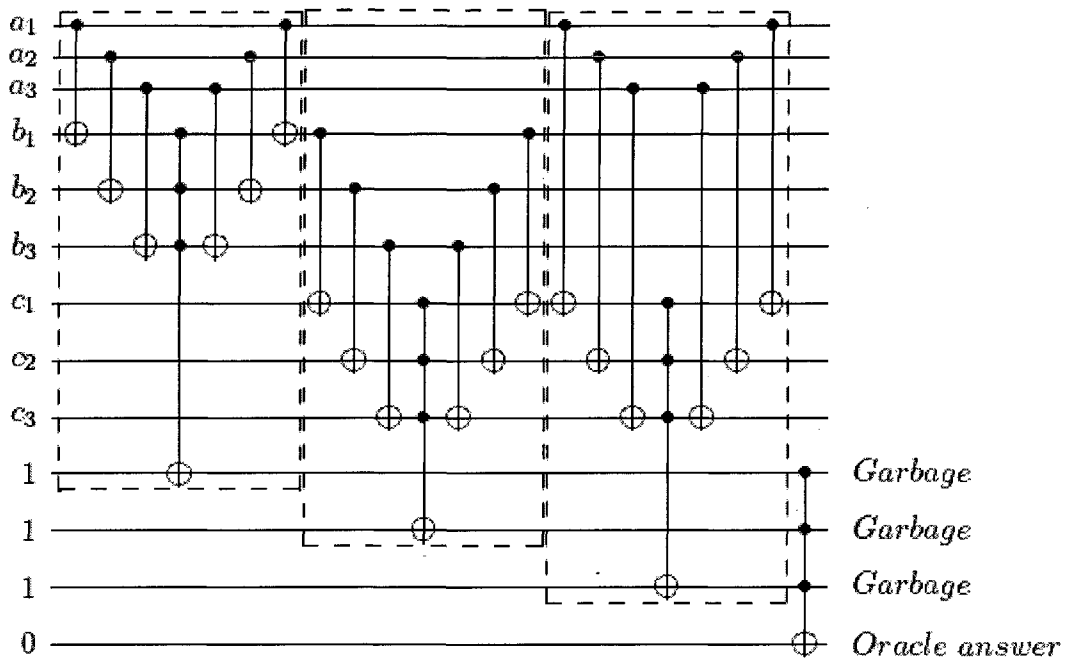


Figure 3.3.7: Quantum array realized for the classical oracle from Figure 3.3.6. Observe three additional ancilla bits. There is 4 ancilla qubits here and this is not taking into account additional ancilla qubits necessary for realization of the four 4×4 Toffoli gates.

Figure 3.3.7 illustrates three quantum comparators ($a \neq b$), ($b \neq c$), ($a \neq c$) quantumly ANDed to give the minimum solution of its classical counterpart in Figure 3.3.6. Mirror gates are added to preserve the original values in qubits b_1, b_2, b_3, c_1, c_2 and c_3. This oracle requires four ancilla bits but the lower bound is only one ancilla bit. The circuit with one ancilla bit would be however very expensive.

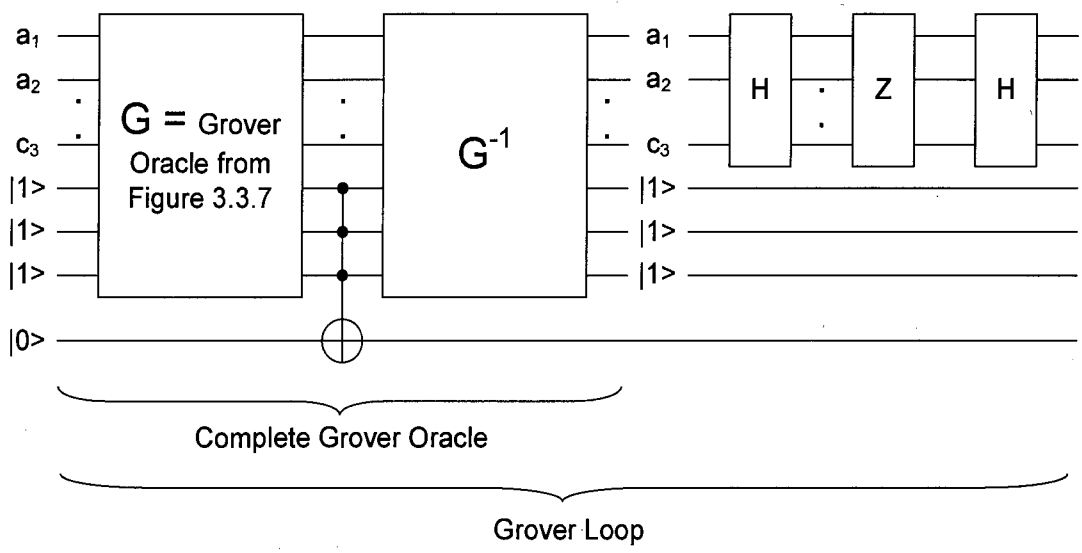


Figure 3.3.8: Complete Grover Loop for the simple graph coloring problem.

3.4. Selected Basic Concepts and Formalisms for Classical, Reversible and Quantum Circuits Analysis and Synthesis.

In this section we present briefly selected notions that will be used in the next chapters.

3.4.1. Tensor products.

To explain better quantum simulation used in calculating all fitness functions for quantum circuits, we have to go deeper to the analysis of quantum circuits.

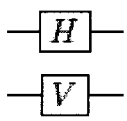


Figure 3.4.1.1: Parallel connection of gates H and V.

Let us calculate for instance the unitary matrix of the circuit from Figure 3.4.1.1 above. We use Kronecker operation as follows:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\otimes\frac{1}{2}\begin{bmatrix}1+i & 1-i\\ 1-i & 1+i\end{bmatrix}$$

It is also called the Tensor Product. It can be illustrated on symbolic values as in Equation 3.4.1.1 below:

$$\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\otimes\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}=\begin{bmatrix}a_{11}\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix} & a_{12}\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}\\ a_{21}\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix} & a_{22}\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}\end{bmatrix} \quad \textit{(Equation 3.4.1.1)}$$

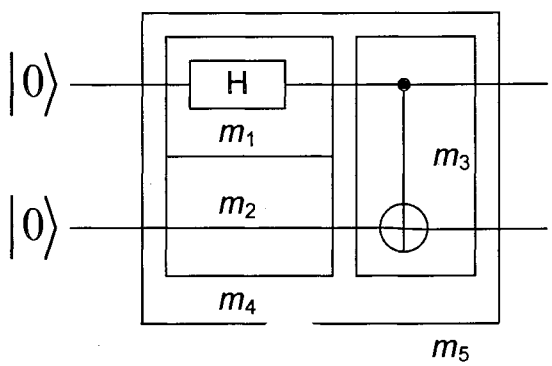


Figure 3.4.1.2: Decomposition of the famous Einstein-Podolsky-Rosen (EPR) circuit (that produces entanglement) to parallel and serial blocks in order to calculate its unitary matrix.

The decomposition of the entire circuit for EPR entanglement is shown in Figure 3.4.1.2. The formula for final unitary matrix is given in Equation 3.4.1.4 below:

$$m_3 \cdot \underbrace{(m_1 \otimes m_2)}_{m_4} = m_5$$

Figure 3.4.1.3: Symbolic Decomposition of the EPR circuit to matrix operations corresponding to the parallel and serial blocks.

The calculations are performed step-by-step as in Equations 3.4.1.2 -3.4.1.4 below:

$$m_4 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes I$$

(Equation 3.4.1.2)

$$m_4 = m_1 \otimes m_2 = \begin{bmatrix} \frac{1}{\sqrt{2}}I & \frac{1}{\sqrt{2}}I \\ \frac{1}{\sqrt{2}}I & -\frac{1}{\sqrt{2}}I \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

(Equation 3.4.1.3)

$$m_5 = m_3 * m_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

(Equation 3.4.1.4)

Now we can introduce in a simple way the Dirac and Heisenberg notations and their mutual links:

Dirac Notation for the initial state: $|0\rangle \otimes |0\rangle = |00\rangle$

Corresponding to it Heisenberg Notation: $\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

We calculate the final output state for the input state $|00\rangle$. This is shown in Equation 3.4.1.5 below:

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(Equation 3.4.1.5)

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}|00\rangle + 0|01\rangle + 0|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

$$= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Observe that both Dirac and Heisenberg notations are useful and we have to be able to go from one of them to another one. The above type of calculations is done in any circuit analysis, for instance in every quantum simulator and while calculating fitness functions in genetic and similar algorithms for quantum circuit synthesis. Here we illustrate the Matlab simulation 3.4.1 of EPR Circuit's (Figure 3.4.1.2) output which verified our above mathematical analysis as well. Which clearly shows the counterintuitive and revolutionary property of the EPR circuit's Entanglement.

m5_00 =	*m5_01 =*	*m5_10 =*	*m5_11 =*
0.7071	*0*	*0.7071*	*0*
0	*0.7071*	*0*	*0.7071*
0	*0.7071*	*0*	*-0.7071*
0.7071	*0*	*-0.7071*	*0*

Simulation 3.4.1: Matlab simulation for Figure 3.4.1.2.

3.4.2. Permutative notation for permutative circuits.

Algorithms such as MMD [Maslov05, Maslov05a, Maslov05b, Maslov06] use simple permutative notation to represent permutative circuits. This notation can be used in both the group-theory based algorithms and in the enumerative or evolutionary algorithms. This notation cannot be used for quantum circuits represented by unitary but non-permutative matrices. The example of permutative notation is shown below:

[0, 3, 1, 2, 4, 6, 5, 7]

Its corresponding truth table is shown in Table 3.4.2.1

a	b	c	A	B	C
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
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 3.4.2.1: Truth table for reversible function [0, 3, 1, 2, 4, 6, 5, 7]. It shows that index 0 (000) is mapped to value 0 (000), index 1(001) is mapped to value 3(011) and so on.

3.4.3. Recursive use of Shannon Expansions to create trees.

3.4.3.1. Shannon expansions.

Some new quantum circuit synthesis methods that we created are based on expansions. All expansions historically started from the famous Shannon expansion, illustrated by Equation 3.4.3.1.1 below:

Example 3.4.3.1.1:

$$\begin{aligned} F(a,b,c,d) &= \bar{a}\cdot F\big|_{a=0}(a,b,c,d) + a\cdot F\big|_{a=1}(a,b,c,d) \\ &= \bar{a}\cdot F_0(a,b,c,d) + a F_1(a,b,c,d) \\ &= \bar{a} F_0(a,b,c,d) \oplus F_1(a,b,c,d) \end{aligned} \qquad \textit{(Equation 3.4.3.1.1)}$$

To illustrate a practical expansion for a function, let us assume:

$$F = a\bar{b} + \bar{a}c + bcd + a\bar{c}d$$

We will calculate Shannon expansions step by step:

$$F_{\bar{a}} = F(a,b,c,d)\big|_{a=0} = 0\cdot\bar{b} + \bar{0}\cdot c + bcd + 0\cdot\bar{c}d$$
$$= c + bcd = c$$
$$F_a = F(a,b,c,d)\big|_{a=1} = 1\cdot\bar{b} + \bar{1}\cdot c + bcd + 1\cdot\bar{c}d$$
$$= \bar{b} + 0\cdot c + bcd + \bar{c}d = \bar{b} + bcd + \bar{c}d$$

Shanon Expansion in classical logic is implemented with a standard Multiplexer. This expansion can be also used in Reversible and Quantum Logic and is the base of Davio expansions and new expansions introduced in chapters 7, 8, 9.

3.4.3.2. Shannon Expansion using Multiplexer

Shannon Expansion can be illustrated using a classical multiplexer, as shown in Figure 3.4.3.2.1 below. The input to data input 0 is the negative cofactor with respect to the (control) variable a, and the input to data input 1 of the mutiplexer is the positive cofactor of function F with respect to its input variable a. The special easy case of this expansion is illustrated in Figure 3.4.3.2.2.

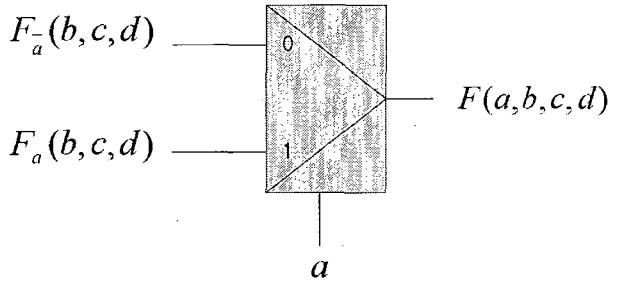


Figure 3.4.3.2.1: General representation of Shannon Expansion of Boolean function F(a,b,c,d) using a classical multiplexer. The data inputs show the cofactors with respect to the control variable a.

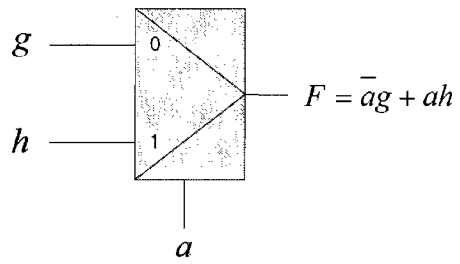


Figure 3.4.3.2.2: The multiplexer and the formula from its Shannon Expansion for simple function $F = \bar{a}\, g + ah = \bar{a}\, g \oplus ah$.

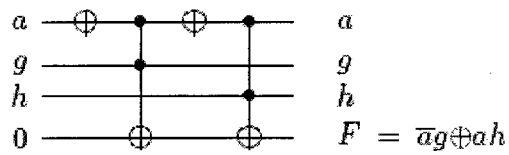


Figure 3.4.3.2.3: The quantum array for the multiplexer of Shannon Expansion from Figure 3.4.3.2.2. Functions g and h on outputs can be either reused in next stages of the quantum array or they will become garbage.

3.4.3.3. Recursive Shannon Expansions create a Tree of Multiplexers

Given is function G

$$G(a,b,c,d) = abc + acd + \bar{a}b + \bar{c}d$$

We will calculate recursively expansions of function G in some order of variables a, b, c, d and next we will draw the tree of these expansions. We select a as the first

expansion variable and we calculate negative cofactor $G_{\bar{a}}$ and positive cofactor G_a for this variable:

$$G_{\bar{a}} = 0 \cdot bc + 0 \cdot cd + 1 \cdot b + \bar{c}\bar{d} = b + \bar{c}\bar{d}$$
$$G_a = 1 \cdot bc + 1 \cdot cd + 0 \cdot b + \bar{c}\bar{d} = bc + cd + \bar{c}\bar{d}$$

Then expanding new functions H (b, c, d) and F (b, c, d) for variable b we get the following sub-functions.

$$H_{\bar{b}} = 0 + \bar{c}\bar{d} = \bar{c}\bar{d} = J(c,d)$$
$$H_b = 1 + \bar{c}\bar{d} = 1$$
$$F_{\bar{b}} = 0 \cdot c + d = d$$
$$F_b = 1 \cdot c + d = c + d = I(c,d)$$

Then expanding new functions J(c, d) and I(c, d) for variable c we get the following expansions.

$$J_{\bar{c}} = 1 \cdot d = d$$
$$J_c = 0 \cdot d = 0$$
$$I_{\bar{c}} = 0 + d = d$$
$$I_c = 1 + d = 1$$

Based on recursion of the above expansions we can draw now the classical tree of multiplexers, Figure 3.4.3.3.1.

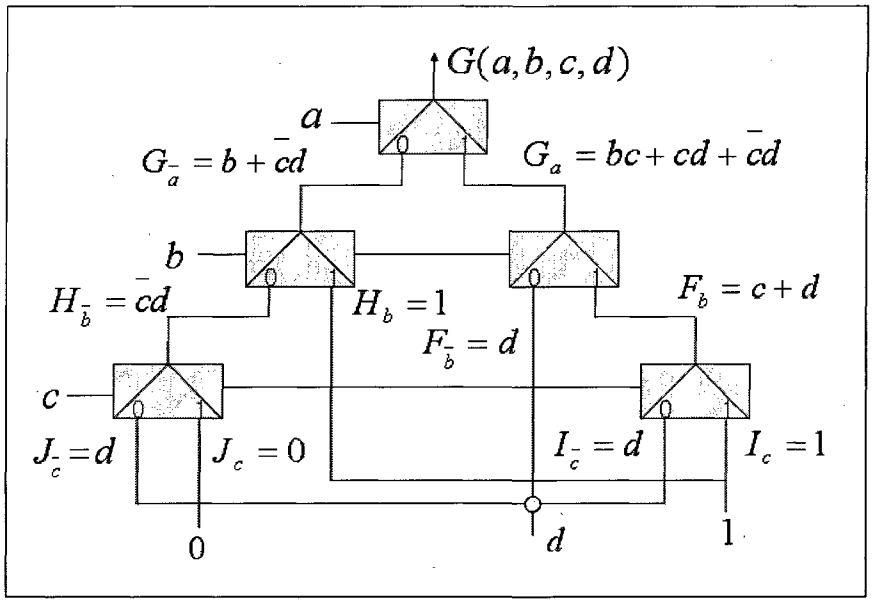


Figure 3.4.3.3.1: Multiplexer based realization of a classical circuit for function G (a, b, c, d).

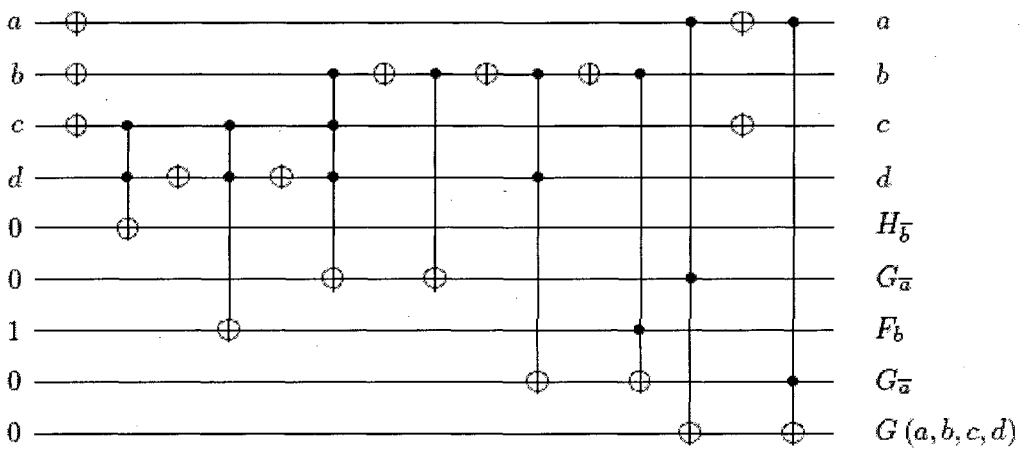


Figure 3.4.3.3.2: Quantum array for the classical circuit from Figure 3.4.3.3.1.

Finally the quantum array corresponding to the tree of multiplexers is shown in Figure 3.4.3.3.2. Please observe many ancilla qubits. Our methods will attempt at reducing

the number of these ancilla qubits. This way, the expansions and classical multiplexers can be used in quantum arrays of oracles. The concept of classical multiplexer will be next transformed to the new concept of a quantum multiplexer that plays a critical role in quantum circuits.

3.4.4. Generalized control Quantum gates with other than AND controlling functions.

Toffoli is the most important quantum gate, but we can observe that similar gates can be created with the same or even lower costs. The importance of Toffoli gate is perhaps only historical and didactical, not technical. While the Toffoli gate realizes a function of AND of its controls, other controlling functions can be realized. Figure 3.4.4.1 presents a reversible function which uses control OR of inputs a, b instead of control AND of inputs a, b. Can we realize this function in quantum? At what realization cost?

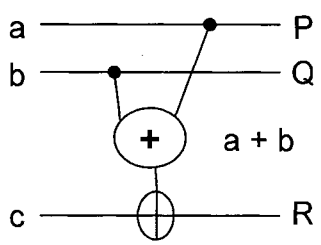


Figure 3.4.4.1: Quantum gate controlled by a + b. We have P = a, Q = b, R = (a+b)⊕ c.

Using KMap-based synthesis methods outlined in this chapter one can find the realization of function (a+b) $\oplus$ c from Figure 3.4.4.2.

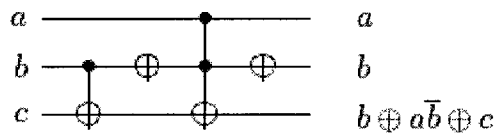


Figure 3.4.4.2: A non-optimal realization of (a+b)$\oplus$ c. It uses a complete Toffoli gate as its part.

We can realize this gate much cheaper using the CV/CV† approach originated by Barenco and much extended in chapter 7 of this thesis. The quantum circuit that realizes the function realized by the symbol-level circuit from Figure 3.4.4.1 is presented in Figure 3.4.4.3.

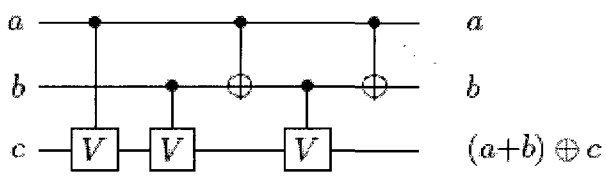


Figure 3.4.4.3: The circuit with CV and CNOT gates that realizes inexpensively the same function as the circuit from Figure 3.4.4.1.

Larger circuits using CV/CV† gates can be also built using the exhaustive reachability method developed by Hung, Song, Yang and Perkowski [Hung04, Hung06]. For instance the circuit from Figure 3.4.4.4 realizes function *F = majority(a, b, c) EXOR d.* This circuit was not invented in [Hung04, Hung06]. Observe that this circuit uses only

truly quantum gates (2×2 primitives) and not some abstract macros with more than 2 inputs. This and similar circuits can be analyzed based on the quantum transformation rules from Figure 3.4.4.5a. Symbolic analysis of this circuit is shown in Figure 3.4.4.5b. If realized only from Toffoli macros, the circuit would be much more expensive, as shown in Figure 3.4.4.6.

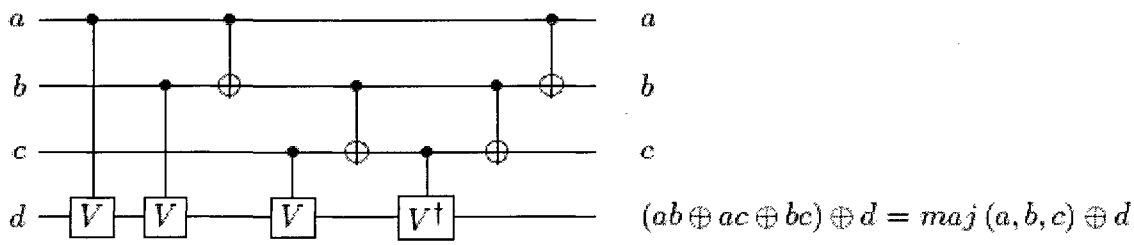


*Figure 3.4.4.4: A circuit that uses only 2*2 truly quantum gates to realize an otherwise complex function maj (a, b, c) ⊕ d if realized with Toffoli gates.*

$$V = \frac{1}{2}\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

$$V = \sqrt{NOT}$$

$$V \cdot V = \sqrt{NOT}\ \sqrt{NOT}$$

$$= \sqrt{(NOT)^2} = NOT$$

$$V.V^{\dagger} = V^{\dagger}.V = I\,;\ V^{\dagger}.\ V^{\dagger} = NOT$$

Figure 3.4.4.5a: Basic quantum algebra rules for CV and CV† gates.

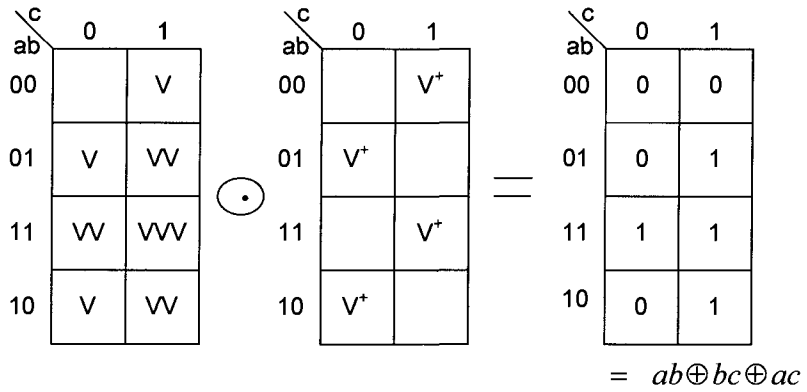


Figure 3.4.4.5b: Symbolic graphical analysis of the circuit from Figure 3.4.4.4. The graphical method of composing Quantum KMaps (QMaps) shown here is the base of my methods presented in chapter 7. Symbol $\odot$ *stands for composing symbolic QMaps.*

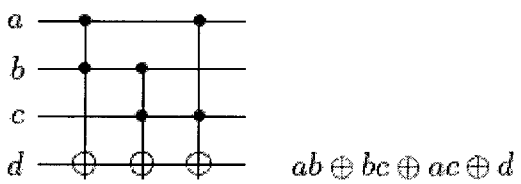


Figure 3.4.4.6: A non-optimal structure for the circuit from Figure 3.4.4.4. As we see this circuit is much more expensive than the circuit using CV/CV$^+$ gates (from Figure 3.4.4.4) because it uses the non-directly –quantum-realizable 3×3 Toffoli gates of high quantum realization cost each.

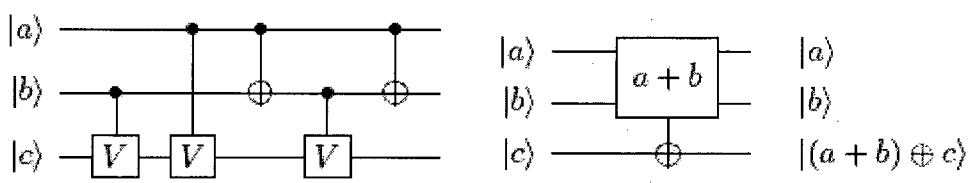


Figure 3.4.4.7: 000. Realization of function $|c\rangle = |(a+b)\oplus c\rangle$ *using only 2-qubit quantum primitives.*

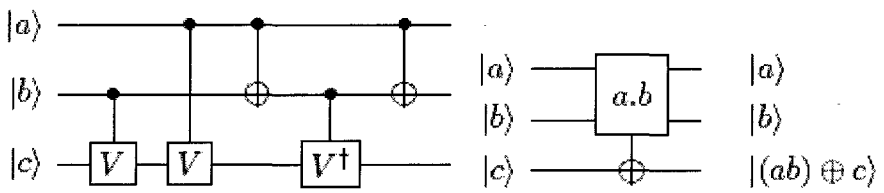


Figure 3.4.4.8: 001. Realization of standard Toffoli gate.

Figures 3.4.4.1 and Figure 3.4.4.4 presented thus two powerful generalizations of CV/CV† based Toffoli gate. They were not known to Barenco [Barenco95] and Smolin [Smolin96]. How far can we go in using the quantum primitives CV and CV† to create powerful permutative macros? This is answered in Figures 3.4.4.7 – 3.4.4.15.

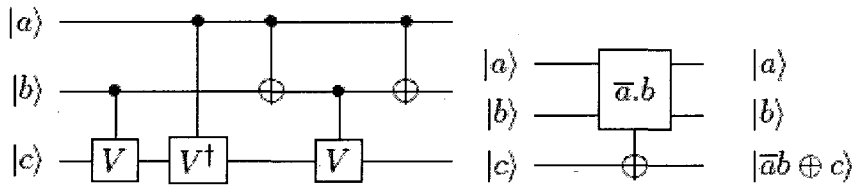


Figure 3.4.4.9: 010 Realization of function $|c\rangle = |\bar{a}b \oplus c\rangle$ using only 2-qubit quantum primitives.

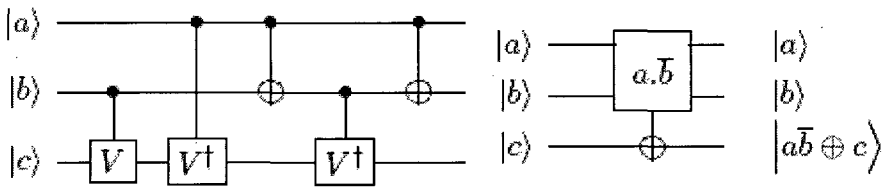


Figure 3.4.4.10: 011 Realization of function $|c\rangle = |a\bar{b} \oplus c\rangle$ using only 2-qubit quantum primitives.

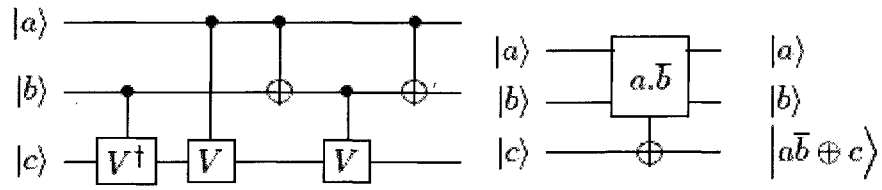


Figure 3.4.4.11: 100 Another realization of function $|c\rangle = |a\bar{b} \oplus c\rangle$ *using only 2-qubit quantum primitives.*

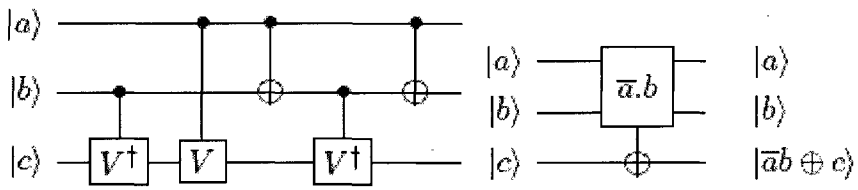


Figure 3.4.4.12: 101 Another realization of function $|c\rangle = |\bar{a}b \oplus c\rangle$ *using only 2-qubit quantum primitives.*

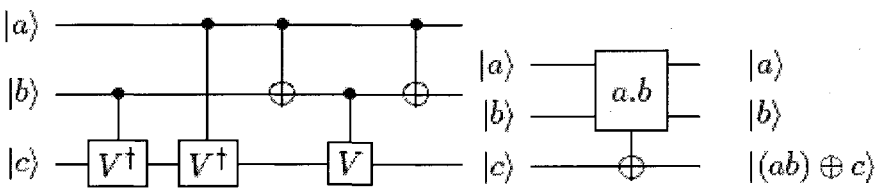


Figure 3.4.4.13: 110 Another realization of standard Toffoli gate.

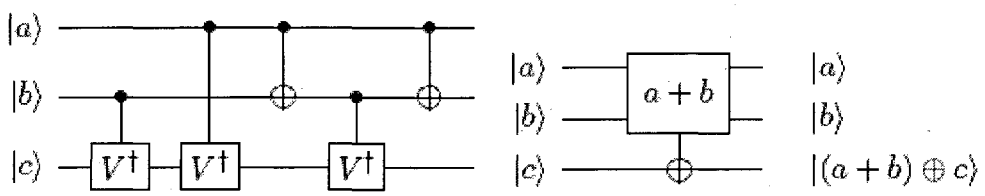


Figure 3.4.4.14: 111 Another realization of function $|c\rangle = |(a+b) \oplus c\rangle$ *using only 2-qubit quantum primitives.*

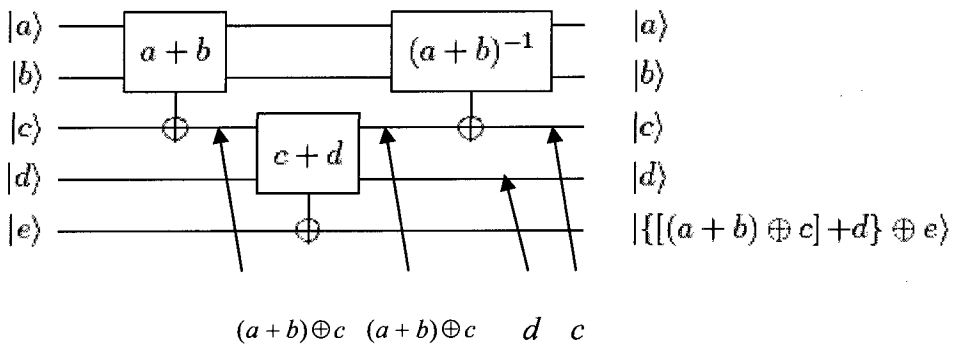


Figure 3.4.4.15: Example of cascading new gates from Figure 3.4.4.7 – 3.4.4.14.

Observe that by permutating CV and $CV^{\dagger}$ gates, one can create many useful gates as shown in Figures 3.4.4.7 – 3.4.4.14. Though we find repeating gates of the same functionality with different circuits like Figure 3.4.4.9 and Figure 3.4.4.12. Also Figure 3.4.4.10 and Figure 3.4.4.11. However, this repetition does not lead to any disadvantage, besides, we can use this as well. It is representations-two ways to realize the same functionality. The mirror gates can be used (Figure 3.4.4.15) to restore original values of input variables a, b, c, d while creating larger gates from these primitives (as useful in oracles). Also, by removing the right most CNOT gate new variants of Peres gates are created. Thus creating Peres families that are larger than Toffoli families because in Peres families many linear functions are returned in all upper bits instead of only the original inputs. Our methods from chapters 7 - 9 will extend these ideas.

3.4.5. Controlled-root-of-NOT gates.

G gate is the square root of square root of NOT. Obviously the tautological transformations from Figure 3.4.5.1 below apply to this gate.

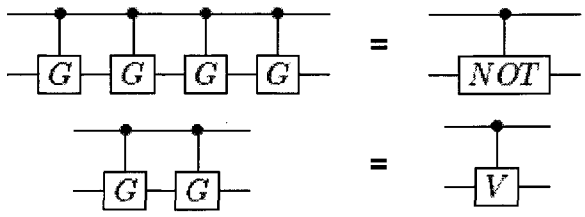


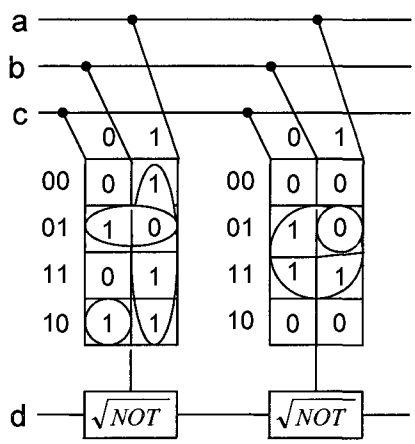
Figure 3.4.5.1: Realization of Controlled-NOT and Controlled-V gate from Controlled-G gates.

Similar transformations can be created for arbitrary root-gates-of NOT gates; $NOT^{1/k}$, k = 4, 5, 6.....

Using a combination of new methods from the thesis all circuits derived by Barenco using V and G in his famous paper [Barenco95] can be created as just few special cases.

3.4.6. Controlling V gates based on arbitrary controls.

Figure 3.4.6.1 presents a circuit in which the Controlled-V gate is controlled by an arbitrary function. This circuit concept generalizes the standard Controlled-V gate.



$[c \oplus \bar{a}\, b \oplus a \bar{b}\, \bar{c}]CONTROL\sqrt{NOT} \odot [b \oplus \bar{a}\, b\, c]CONTROL\ \sqrt{NOT}$

Figure 3.4.6.1: Controlled- V gates with arbitary controlling functions.

The analysis of the circuit from Figure 3.4.6.1 is shown in Quantum Kmap from Figure 3.4.6.2.

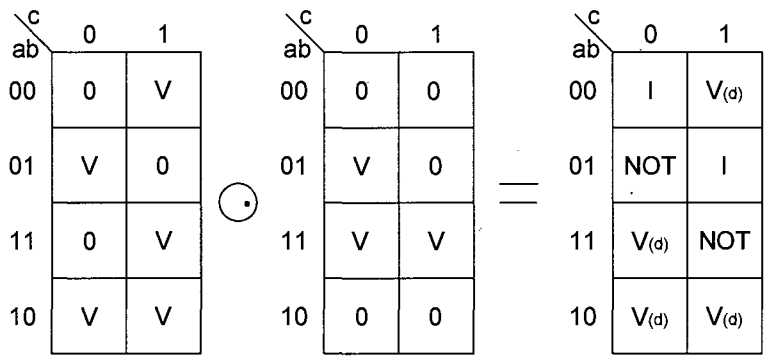


Figure 3.4.6.2: QMap Analysis of the circuit using Controlled-V(Controlled-$\sqrt{NOT}$) gates with arbitary controlling functions from Figure 3.4.6.1. Operator $\odot$ means composition. I is the identity transformation.

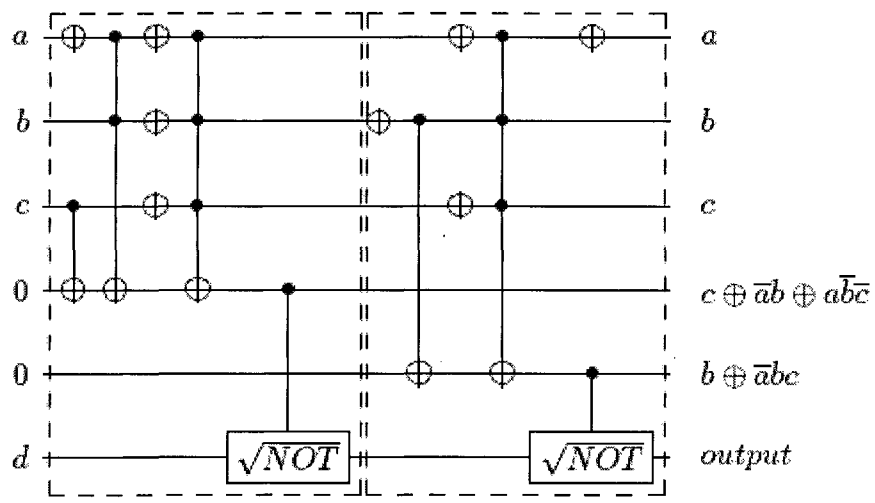


Figure 3.4.6.3: Quantum circuit using Controlled-V(Controlled-$\sqrt{NOT}$) gates with arbitary controlling functions from Figure 3.4.6.2.

The (non-optimized) realization of the circuit from Figure 3.4.6.1 using quantum array is shown in Figure 3.4.6.3.

(a)

T1

c / ab	0	1
00	I	I
01	I	I
11	V	V
10	V	V

T2

c / ab	0	1
00	I	I
01	V	V
11	NOT	NOT
10	V	V

T3

c / ab	0	1
00	I	V
01	V	V.V
11	NOT	NOT.V
10	V	V

T4

c / ab	0	1
00	I	V.V+
01	V.V+	NOT
11	NOT	NOT.V.V+
10	I	NOT

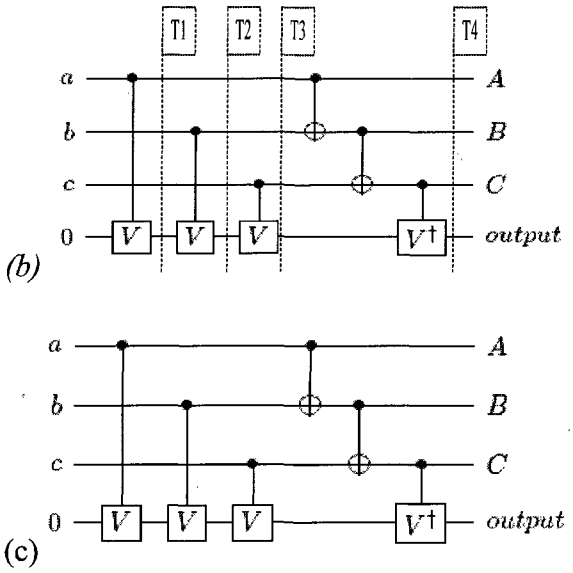


Figure 3.4.6.4: Another example of Controlled- V gates with arbitrary controlling functions (linear in this case). (a) Quantum QMap analysis of the circuit from Figure 3.4.6.4b.(b) Quantum Circuit analyzed from left to right in Figure 3.4.6.4a., (c) Quantum circuit as in Figure 3.4.6.4b without analysis stages T_i, i = 1, ...4. Inputs b, c are not restored.

Another circuit of this type is shown in Figure 3.4.6.4. In addition, the analysis of this circuit using quantum truth table is shown in Figure 3.4.6.5. The quantum states are shown for the lowest (output) qubit in points T1, T2, T3 and T4 from the quantum array from Figure 3.4.6.4, respectively.

a b c	T1	T2	T3	T4
0 0 0	I	I	I	I
0 0 1	I	I	V	$V \cdot V^+ = I$
0 1 0	I	V	V	$V \cdot V^+ = I$
0 1 1	I	V	$V^2 = N$	N
1 0 0	V	V	V	I
1 0 1	V	V	N	N
1 1 0	V	N	N	N
1 1 1	V	N	N·V	$NV \cdot V^+ = N$

Figure 3.4.6.5: Analysis of several functions from cascade (Figure 3.4.6.4) with a single truth table. This method of analysis is more convenient in some cases than the analysis method based on many Quantum QMaps.

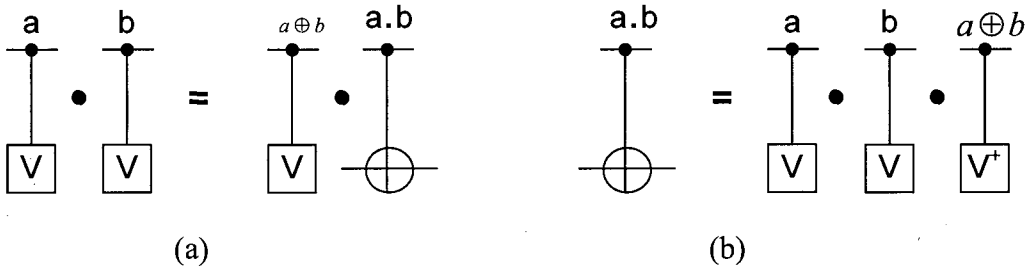


Figure 3.4.6.6: Graphical Illustration of the general algebra rules for controlling quantum gates by Boolean variables.

We invented a set of general graphical transformations, which we can use as general algebra of controlled quantum gates. Say we have the control like in Figure 3.4.6.6 (a), where binary qubits a and b are controlling V gates . It is equivalent to qubit a ⊕ b controlling V, and composed with qubit a • b (AND (a, b)) controlling the NOT gate (CNOT). This is a very useful identity, which we can use in synthesis later on. From Figure 3.4.6.6(a) we can derive Figure 3.4.6.6(b). This is very useful, this is a better way of explaining controlled circuits in the form of a new algebra. We can say that this

kind of transformations is related to analysis of circuits, which is next very useful in our synthesis methods for Quantum Circuits.

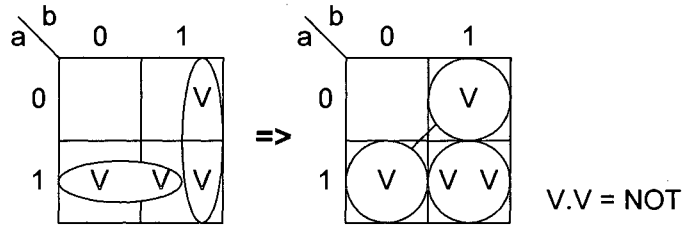


Figure 3.4.6.7: QKMap based analysis of Figure 3.4.6.6a.

QMap interpretation of the rule from Figure 3.4.6.6a is given in Figure 3.4.6.7. Similarly, the QMap interpretation of the rule from Figure 3.5.6.6b is presented in Figure 3.4.6.8.

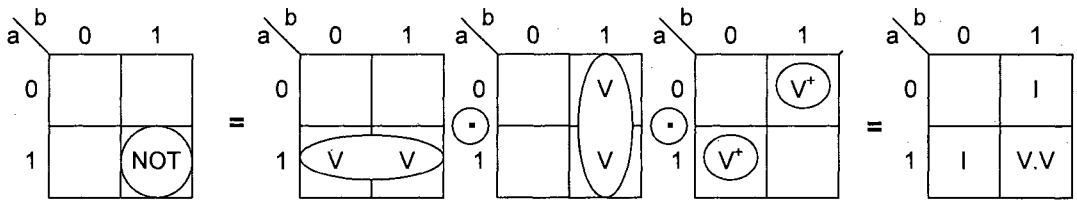


Figure 3.4.6.8: Presents QKMap analysis of Figure 3.4.6.6b.

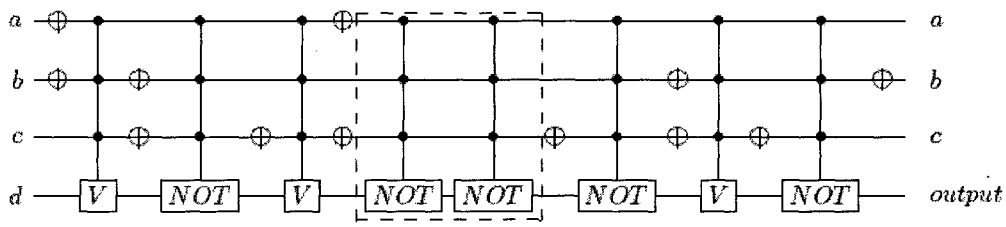


Figure 3.4.6.9: The minimization that can be applied on the gate level. Here two NOT gates can be cancelled. More optimizing transformations can be next extended.

The Figure 3.4.6.1 can be transformed to the circuit from Figure 3.4.6.9. Template matching transformations [Miller03] can be next used iteratively on this circuit to simplify it.

3.4.7. Universal 3 qubit circuits.

A new approach to realize arbitrary 3-qubit circuits is shown in Figure 3.4.7.1. Analyzing every possible combination of control signals a and b we can verify that the schematics on the left and on the right of Figure 3.4.7.1 are equivalent in the sense of having their unitary matrices equal.

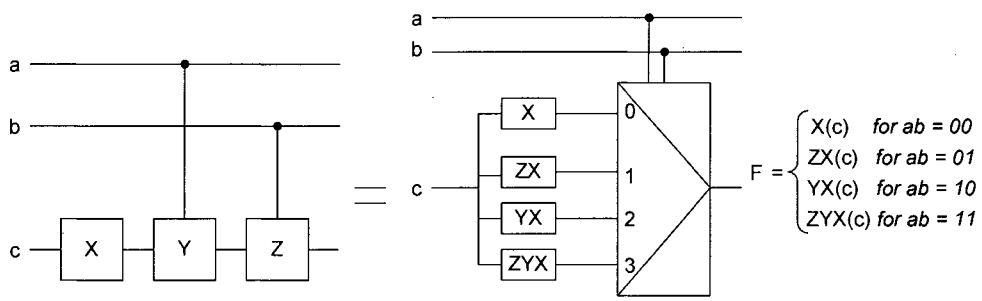


Figure 3.4.7.1: Quantum Circuit from controlled gates versus equivalent to it Quantum Multiplexer Circuit.

The transformation at the right side of Figure 3.4.7.1 shows that the Quantum multiplexer implemented using operators X, ZX, YX, ZYX as data is equivalent to the circuit from controlled gates at the left. This can be verified by multiplying corresponding symbolic unitary matrices. Several similar transformations exist.

3.5. Search and Optimization.

3.5.1. Evolutionary, Search and Quantum Search approaches to Synthesize Quantum Circuits from the above-introduced gates and circuits

Much of my research conducted so far and described in this thesis was to develop a general-purpose algorithmic approaches that would automatically design application specific quantum algorithms for few selected classes of binary logic synthesis and minimization problems. They are included in chapters 7, 8 and 9 of this thesis. However, in order to understand these approaches, sufficient background on classes of functions and design methods for them will be first necessary. Why we believe these algorithms will be better than the approaches used so far?

When I learned about the concepts of evolvable hardware and machine learning, I wanted to make my methods for quantum synthesis to be very general and applicable to logic synthesis, minimization, Data Mining, Knowledge Discovery, and Evolvable Hardware.

Several other new approaches were created in the past at PSU and elsewhere to utilize search and evolutionary techniques for both logic synthesis and circuit minimization of AND/EXOR circuits. Initially developed by Karen Dill [Dill97, Dill97a, Dill97b, Dill97c, Dill98, Dill01] for a single purpose, rather than broadly applicable to binary

logic design and minimization problems some algorithms were viewed as initial trials for the biology inspired methodologies. The results of Karen Dill, Martin Lukac and Normen Giesecke as well as other researchers (about evolutionary, Particle Swarm Optimization (PSO), Bacteria Foraging (BF) methods and cultural, social memetic algorithms) have been critically analyzed by me and new approaches have been thus developed in this dissertation and proved to be better by the experimental software results.

The first design in my research involved the application of various approaches for the minimization of Generalized Reed-Muller (GRM) logic forms. As may be recalled, the GRM equation type is a general, canonical expression of the Exclusive-Or Sum-of-Products (ESOP) type, in which for every subset of input variables there exists not more than one term with arbitrary polarities of all variables. This AND-EXOR implementation has been demonstrated to be economical, generally requiring fewer gates and connections than that of other variants of AND-EXOR logic such as particularly PPRM and FPRM. GRM logic is also highly testable, making it desirable for quantum designs. Research from [Dill97] used standard Darwinian and Lamarckian evolution [Dill01] as a model from which logic minimization algorithms are determined. To date, the few developed exact minimization algorithms have required nearly exhaustive searches on standard computers and are quite time consuming. We found this model insufficient and thus the ideas of search and

quantum search were added and combined with the evolutionary methods. The goal of using our new approaches for AND/EXOR logic in this dissertation was to create non-exact heuristic minimization techniques that would constitute an improvement in the quality for the optimizations produced by the heuristic (rule-based) methods known from the literature. Moreover, the minimization methods developed in this dissertation are applicable to both single-output and multiple-output permutative quantum circuits.

For completely specified data, the GRM equation form has been proven difficult to minimize, as no exact minimization method (other than a nearly exhaustive search) has been devised. For instance, Miller and Thomson [Miller94a] give an exhaustive search algorithm for the FPRM form [Miller94b, Drechsler99]. Exhaustive search methods on classical computers are time consuming, making an effective, and high-quality, approximate minimization method very attractive. On the other hand, the exhaustive methods are of interest in Grover-like quantum computing where the efficiency is not a problem to be practically considered, since such computers simply do not exist. The new concept with its mathematical proof is however theoretically interesting. Thus we created such quantum algorithms in chapter 15.

Several variants of Genetic Algorithms and Genetic Programming were used at PSU to minimize FPRM circuits [Dill97a, Dill97b, Dill97c] and various types of reversible

circuits with general structures [Giesecke06, Giesecke07, Lukac02, Lukac04, Lukac05, Khan03, Khan05a, Khan05b]. For instance, several attempts were made to develop a purely evolutionary (i.e., GA with no human-designed heuristics) approach to the minimization of GRM forms. As no application-specific knowledge was incorporated to these methods [Dill97b, Dill98, Dill01], the results were remarkable as they compared favorably with that of the heuristic algorithms designed by human experts [Debnath95, Debnath96]. On the other hand, for some functions, Sasao and Debnath [Debnath98] found better solutions using heuristic knowledge-based algorithm, which showed that the evolutionary approach should be possibly equipped with more human-like knowledge and/or human intervention in the automatic solution process. The first approach to minimize incompletely specified functions has been also developed by the PSU team [Zheng95, Dill97b, Dill98, Dill01]; the GRMin software was created. But this was only done for small, single-output functions. Although Debnath and Sasao [Debnath96, Debnath98] developed a successful heuristic for GRM minimization, capable of handling functions with a large number of variables and multi-outputs, their software (not available in public domain) was applicable only to completely specified functions. Finally, I develop in this thesis the ECPS software culminating the efforts of the PSU team that have started many years ago. I proved experimentally on many benchmarks that this software is better than all previous software.

Few authors [Green91, Mckenzie93, Varma91, Riege92, Zilic02] have considered the problem of Positive-polarity Reed-Muller (PPRM) form minimization for single-output incompletely specified functions. However, with the exception of work by Zilic and Vranesic [Zilic02], the algorithms are very inefficient for functions that have a large number of don't cares, as the algorithm complexity increases with the amount of unspecified data. Moreover, all these algorithms cannot be adapted to the GRM form, which is quite different from that of the PPRM form.

The minimization of incompletely specified functions is well known to be more difficult than the minimization of the completely specified functions. This problem is important also because of its possible applications in Data Mining and Machine Learning. For instance, Chang and Falkowski [Falkowski97] developed a FPRM minimization algorithm for a small percentage of don't cares. On the other hand, Zakrevskij [Zakrevskij95] developed a FPRM minimization algorithm for FPRMs that is efficient only for a very high percentage of don't cares. Similarly, it is most difficult to minimize ESOPs for the incompletely specified functions that have 5 – 95 % don't cares. It can thus be predicted, for GRMs also, that the minimization of few (<5%) or very many (>95%) don't cares is easier than the case of a medium amount of don't cares. The iGRMMIN minimization algorithm [Dill97a, Dill97b] performed well for all categories. The software developed by me for this dissertation performs even better.

As may be recalled, while more restrictive than the Exclusive-Or Sum-of-Products (ESOP) expression, the GRM equation form incorporates the Fixed-Polarity Reed-Muller (FPRM) and Positive-Polarity Reed-Muller (PPRM) forms as its special cases. The GRM is a canonical expression that allows complete freedom as to the polarity selection of each term, but there is at most one product term for every subset of variables.

The new GRM minimizing software is the second application of the GRM form to the synthesis and minimization of incompletely specified data. A multi-strategic approach was taken. Human expertise was combined with the genetic search mechanism, for the development of an efficient problem-solving expert system. The goal of using the Genetic Algorithm for GRM minimization was simply to aid the solution search process for the human-designed logic minimization heuristic.

The results of various algorithms developed in this dissertation are compared with those from [Dill01, Sasao90a, Sasao93, Stankovic97, Lukac07]. My numerical results from this dissertation supersede the previous results from other authors . My results are obtained also for a more general family of structures than GRM. The conceptual and software approaches from my dissertation are also applicable to PPRM forms, FPRM forms, and other canonical forms, as well as to ESOPs, factorized circuits, and

circuits with linear preprocessor and affine circuits thus allowing to compare uniformly many variant designs of a circuit in quantum technologies.

3.5.2. Formalism for Expansions.

Example 3.5.2.1: A GF-PPRM, in GF(2), is generated by the application of the Positive Davio Expansion, (i.e. all literals, $\bar{x}_i$ s have positive polarity). For binary logic using three variables, an example is given.

$f(x_1, x_2, x_3) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus a_4x_1x_2 \oplus a_5x_1x_3 \oplus a_6x_2x_3 \oplus a_7x_1x_2x_3$

Example 3.5.2.2: A GF-GRM, in GF(2), has both positive and negative polarities. For binary logic using three variables, an example is given.

$f(x_1, x_2, x_3) = a_0 \oplus a_1X_1 \oplus a_2X_2 \oplus a_3X_3 \oplus a_4X_1X_2 \oplus a_5X_1X_3 \oplus a_6X_2X_3 \oplus a_7X_1X_2X_3$

Where, X= x or $\bar{x}$

In addition to being the standard Reed-Muller forms, the expressions in Examples 3.5.2.1 and 3.5.2.2 are actually polynomial forms in GF(2) for three variables.

With this background, the Galois Fields from chapter 3 can now be fully related to the Reed-Muller Logic. Most central to the development of Reed-Muller logic forms, the classical Shannon Expansion utilizes a variable polarity separation technique to

represent a function. The Shannon Expansion for a variable x is obtained by splitting the variable into two different polarities, x and $\bar{x}$. The relation between these polarities can be represented as $x = 1 \oplus \bar{x}$. For a binary function $f(x_1, x_2, \ldots, x_n)$ the Shannon Expansion, originally developed by Boole [Boole54, Brown90] is:

$$f(x_1,\ldots,x_n) = \bar{x}_1 f(x_1=0,x_2,x_3,\ldots,x_n) \oplus x_1 f(x_1=1,x_2, x_3,\ldots,x_n)$$

$$f(x_1,\ldots,x_n) = \bar{x} f_0 \oplus x f_1$$

Equation 3.5.2.1: $f(x) = \bar{x} f_0 \oplus x f_1$ (Shannon Expansion)

Relating the Shannon Expansion to a KMap, another perspective can be gained about its application. This gives a visual depiction of how the components "fit" together to make the total function. In Figure 3.5.2.1, a simple KMap is given, with binary values represented by variables, with subscripts labeled for their location.

x \ y	0	1
0	f_{00}	f_{01}
1	f_{10}	f_{11}

Figure 3.5.2.1: Representation of binary cofactors in the Karnaugh map.

In Equation 3.5.2.1 for the Shannon Expansion, f_0 and f_1 are simply rows of the KMap, where x = 0 and 1, respectively. These are given in Figure 3.5.2.2 below.

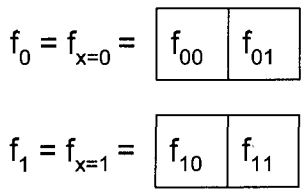


Figure 3.5.2.2: Graphical representation of Shannon expansion for the Karnaugh map from Figure 3.5.2.1.

Starting with the Shannon Expansion, then, the KMap is related as follows in Figure 3.5.2.3.

$f(x) = \bar{x} f_0 \oplus x f_1$ Shannon Expansion, GF(2)

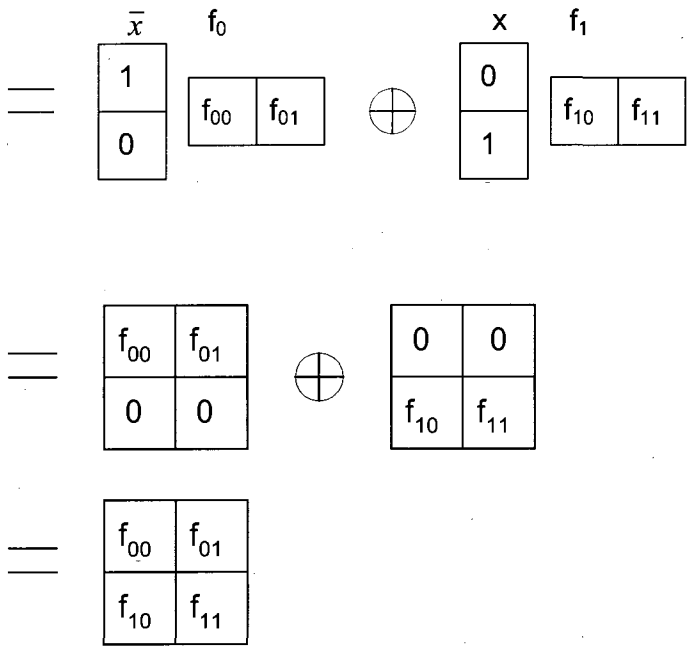


Figure 3.5.2.3: Step-by-step calculation of Shannon expansion with KMap visualization.

The Shannon Expansion shown in algebraic form can also be represented as a decision tree. This is shown in Figure 3.5.2.4.

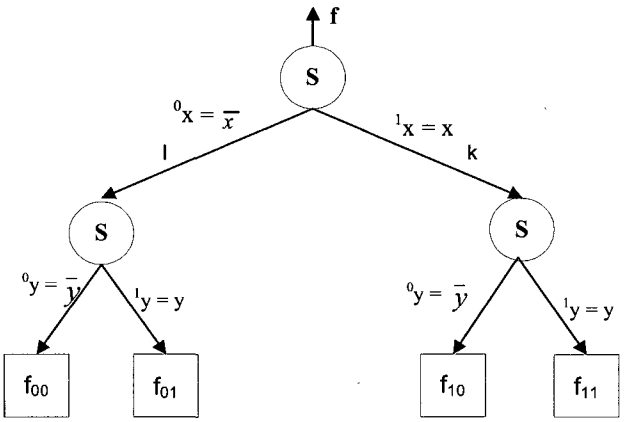


Figure 3.5.1.4: Shannon Tree for binary logic of two variables. Two notations are used for negations, this is useful in mv generalizations of such trees.

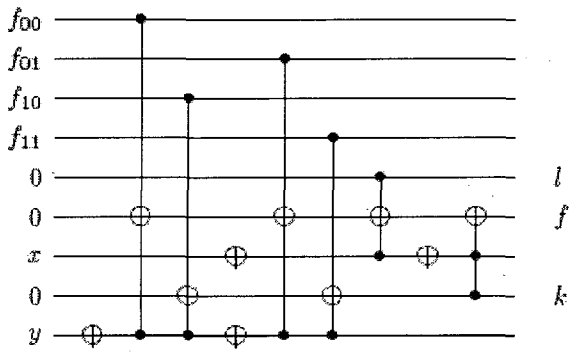


Figure 3.5.2.5: The Quantum array with ancilla bits for nodes l, k, and f drawn directly from the decision diagram from Figure 3.5.2.4.

The Davio Expansions in binary logic are well known and derived from the Shannon Expansion by considering either the positive polarity (x) or negative polarity ($\bar{x}$) of the variable x. (Alternatively, starting from either the Positive Davio or Negative Davio, the Shannon Expansion may be derived). These derivations are shown in Equations

3.5.2.2 and 3.5.2.3 below. The Shannon, Positive Davio, and Negative Davio Expansions may be utilized to derive all possible expansions, to obtain all logic family forms, trees, and decision diagrams.

Derivation of Positive Davio Expansion:

Shannon $f(x) = \bar{x} f_0 \oplus x f_1$

By substituting $\bar{x} = x \oplus 1$

$f(x) = (x \oplus 1) f_0 \oplus x f_1$

$f(x) = x f_0 \oplus f_0 \oplus x f_1$

$f(x) = x(f_0 \oplus f_1) \oplus f_0$

Positive Davio: $f(x) = x(f_0 \oplus f_1) \oplus f_0$

We derive here Davio expansions because they are a fundament of more complex expansions that we will introduce in the sequel.

Equation 3.5.2.2: The Positive Davio Expansion for binary (GF(2)) logic is the following:

$f(x) = x(f_0 \oplus f_1) \oplus f_0$

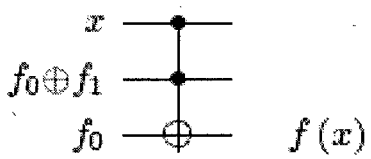


Figure 3.5.2.6: Part of a quantum array to realize the positive Davio expansion, where f_0 and $(f_0 \oplus f_1)$ are functions of remaining variables, which may require ancilla bits.

Example 3.5.2.3:

Let $f = ax \oplus bx \oplus ab = x(a \oplus b) \oplus ab$. Thus the cofactors with respect to variable x are the following :

$$f_{\bar{x}} = f|_{x=0} = ab$$

$$f_x = f|_{x=1} = a \oplus b \oplus ab = (a + b)$$

The Boolean difference is

$$f_0 \oplus f_1 = f_{\bar{x}} \oplus f_x = ab \oplus (a \oplus b \oplus ab) = a \oplus b$$

From this we can draw the quantum array from Figure 3.5.2.7. Observe the mirror circuit to restore variable b.

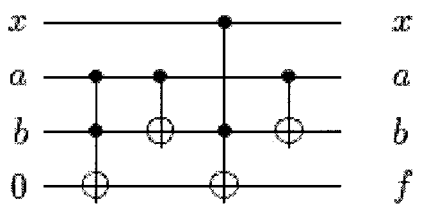


Figure 3.5.2.7: Graphical representation of Positive Davio expansion for function $f = ax \oplus bx \oplus ab = x(a \oplus b) \oplus ab$.

Derivation of Negative Davio Expansion:

Shannon $f(x) = \bar{x}\, f_0 \oplus x f_1$

By substituting $x = \bar{x} \oplus 1$

$$f(x) = \bar{x} f_0 \oplus (\bar{x} \oplus 1) f_1$$

$$f(x) = \bar{x} f_0 \oplus \bar{x} f_1 \oplus f_1$$

$$f(x) = \bar{x}\, (f_0 \oplus f_1) \oplus f_1$$

Negative Davio: $f(x) = \bar{x}\, (f_0 \oplus f_1) \oplus f_1$

Equation 3.5.2.4: The Negative Davio Expansion for binary (GF(2)) logic

$f(x) = \bar{x}\, (f_0 \oplus f_1) \oplus f_1$

The realization of Negative Davio Expansion in quantum array is represented in Figure 3.5.2.8. As the cost of NOT is negligible in all quantum technologies, the negative and positive Davio expansions should be used on equal terms in all synthesis algorithms leading to improved results with respect to the approaches that use only the Positive Davio. This is analogical to the superiority of FPRM over PPRM.

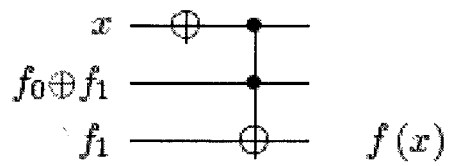


Figure 3.5.2.8: Realization of Negative Davio Expansion.

Expansion trees provide a graphical representation of functional components. As a diagram of cofactors and multipliers (constants), they provide a visual depiction of decision trees, which are a useful tool in deriving the forms of an algebraic family and find also applications in Data Mining.

Expansions such as the Shannon, Positive and Negative Davios can be applied to functions, as a variable separation technique, creating an expansion tree diagram. In expansion tree diagrams, several expansion nodes may be combined, such that each node on a level, corresponding to an expansion variable, has one of the defined expansions. The total function (over the entire tree), in its new form, can then be re-constructed by combining the cofactors and multipliers for each of the branches with the EXOR operation. These methods were used to derive circuits and algorithms from this and next chapters of my thesis.

3.6. Butterfly diagrams for FPRM Forms

This section contains preliminary background discussion on the theory of butterfly diagrams for FPRM forms. Butterfly diagrams are used in transformations and optimizations of AND/EXOR spectral logic [Falkowski97, Falkowski98, Falkowski03] and in our oracles in chapter 15.

A switching function is commonly described in a sum-of-minterms form that is canonic and which in the binary case represents a collection of conjunctive terms joined by a disjunctive operator. As an example, all binary functions of three variables may be expressed in the form

$$F = m_0\overline{x}_1\overline{x}_2\overline{x}_3 + m_1\overline{x}_1\overline{x}_2 x_3 + m_2\overline{x}_1 x_2\overline{x}_3 + m_3\overline{x}_1 x_2 x_3 + m_4 x_1\overline{x}_2\overline{x}_3 + m_5 x_1\overline{x}_2 x_3 + m_6 x_1 x_2\overline{x}_3 + m_7 x_1 x_2 x_3$$

where $m_i \in \{0,1\}$ are commonly referred to as the minterms of the function f. It can be easily proved that every OR operator in this formula can be replaced with EXOR operator because all the minterms are pairwise disjoint. Alternatively, such functions may also be represented as a Reed-Muller expansion of a given polarity using a collection of conjunctive terms joined by the modulo-additive operator as

$$a_0 1 \oplus a_1 \hat{x}_1 \oplus a_2 \hat{x}_2 \oplus a_3 \hat{x}_3 \oplus a_{12} \hat{x}_1 \hat{x}_2 \oplus a_{13} \hat{x}_1 \hat{x}_3 \oplus a_{23} \hat{x}_2 \hat{x}_3 \oplus a_{123} \hat{x}_1 \hat{x}_2 \hat{x}_3$$

where $a_i \in \{0,1\}$ are the FPRM spectral coefficients and $\hat{x}_i$ represents a literal in either complemented or uncomplemented form consistent for all values of i. The particular assignment of polarities of the dependent variables $\hat{x}_i$ leads to the polarity number. For example $x_1 x_2 x_3 \rightarrow 7$, $x_1 \overline{x_2}\, \overline{x_3} \rightarrow 4$, $\overline{x_1}\, x_2 x_3 \rightarrow 3$, etc.

The problem of interest in several classical, reversible and quantum logic synthesis problems is to find the polarity number $P \leq T$ such that a Reed-Muller expansion can be formed where at least T spectral coefficients are zero-valued. The solution of this problem allows for the realization of a FPRM expansion that utilizes no more than T conjunctive operations and is a problem of interest for the logic synthesis and verification community. We discuss here the FPRM case, but similar techniques are used for other canonical AND/EXOR forms such as GRM, GPMPRM [Zeng95] and other group-based forms (Linearly Independent logic expressions).

The two expressions shown previously are both canonical forms (they may be affine functions in a special case) that are related by a linear transformation. This transformation is well-known and is commonly characterized by a linear transformation matrix as the fixed-polarity Reed-Muller transform [Perkowski97, Perkowski97a, Perkowski97c]. The structure of this transformation matrix can be expressed as a Kronecker (or tensor) matrix product where each dependent variable is

represented by a matrix representing a given polarity. As an example, the transformation matrix for the PPRM transformation is given as

$$M_{P1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad M_{Pn} = \bigotimes_{i=1}^{n} M_{Pi.}$$

For an FPRM, the negative polarity matrix used in forming the transformation matrix M_{N1} is used to represent complemented variables and that of M_{P1} is used for positive-polarity variables. As an example the transformation matrix for an FPRM of polarity 5 is formed as

$$M_{N1} \otimes M_{P1} \otimes M_{N1}$$
$$= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Due to the Kronecker product decomposition of an FPRM transformation matrix, the techniques first attributed to [Lee86, Li06] may be used to represent the transformation in the so-called "butterfly" signal flow-graph (also known as a "fast transform") where edges represent multiplicative weights (in this case all weights are unity) and vertices represent additions modulo-2, shown in Figure 3.6.1. Here we also add more details about the butterfly structure with Exor Map and the mapping of coefficient of polarity in KMap for better explaining of our method.

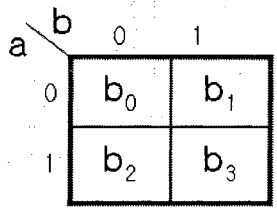


Figure 3.6.1: Coefficients of cells of 2-variable KMap for symbolic transformation.

The symbolic transformation for coefficients from Figure 3.6.1 given as follows:

$$b_0\bar{a}\bar{b} \oplus b_1\bar{a}b \oplus b_2a\bar{b} \oplus b_3ab$$
$$= \underbrace{b_0 \cdot 1}_{c_0} \oplus \underbrace{(b_0 \oplus b_2)a}_{c_2} \oplus \underbrace{(b_0 \oplus b_1)b}_{c_1} \oplus \underbrace{(b_0 \oplus b_1 \oplus b_2 \oplus b_3)ab}_{c_3}$$

From above symbolic transformation we can get the butterfly circuit for this transformation, as shown in Figure 3.6.2 below.

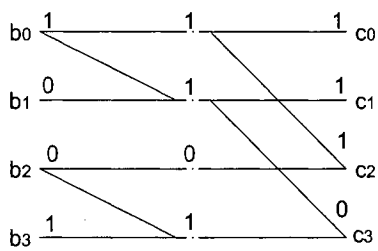
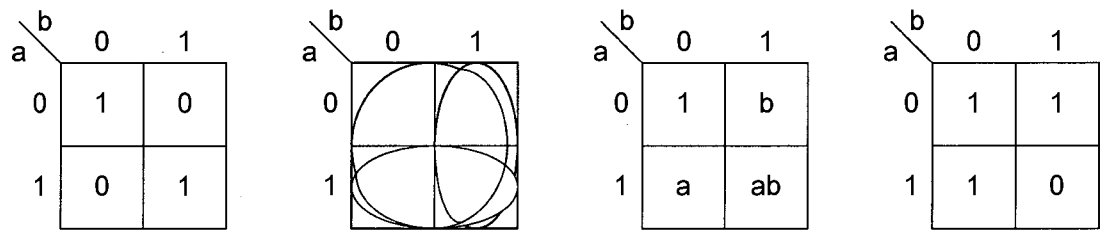


Figure 3.6.2: Butterfly structure for transforming minterms b_i of a Kmap to spectral coefficients c_i of the corresponding PPRM form for two variables.

From the outputs of the butterfly in Figure 3.6.2 we get the symbolic transformation as below:

$$c_0 \oplus c_2 a \oplus c_1 b \oplus c_3 ab$$
$$= 1 \oplus 1 \bullet a \oplus 1 \bullet b \oplus 0 \bullet ab$$
$$= 1 \oplus a \oplus b$$

Now, Figure 3.6.3 explains the symbolic transformation of butterfly structure in Exor Map for positive polarity (a = 1, b = 1) and the mapping of polarity coefficients in the KMap.



(a) (b) (c) (d)

Figure 3.6.3: Conversion of PPRM. a) The KMap of the function being realized, every cell represents the minterm of the function (b) The K-map with the groups selected to realize the function from Figure 3.6.3a, (c) Mapping of variables in product terms of PPRM in Exor Map for positive polarity, (d)Every cell in this positive polarity Exor Map is now not a minterm but a Exor Map coefficient on c_i from the butterfly structure in Figure 3.6.2. Thus $1 \oplus a \oplus b = \overline{a \oplus b} = \bar{a}\bar{b} \oplus ab$ *as in Figure 3.6.3a.*

We will show and explain an Example for FPRM in butterfly structure and its Exor Map for certain fixed polarity of 3 variables (Figure 3.6.4 and Figure 3.6.5).

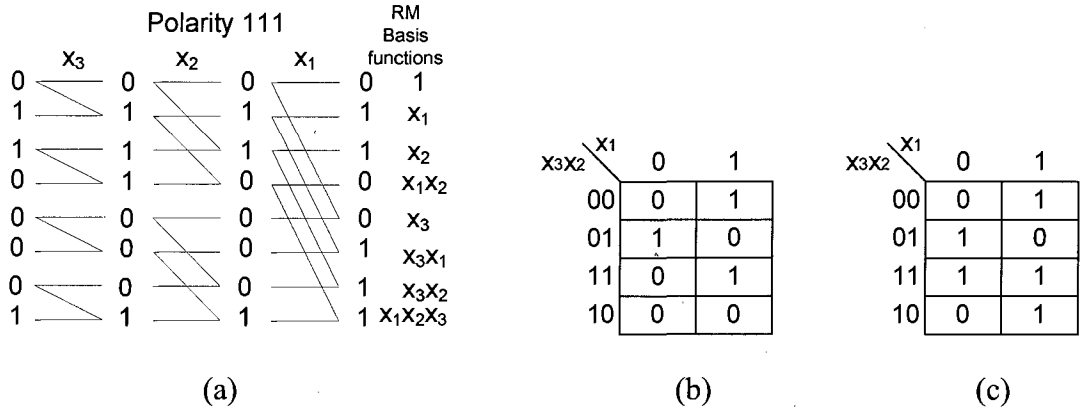


Figure 3.6.4: Conversion from minterms to FPRM with polarity 111 (PPRM). (a) A Butterfly signal flow-graph for the polarity 111 of function F represented by minterms at the left, (b) Karnaugh Map of the minterms, (c) Coefficient mapping in the Exor Map for polarity 111.

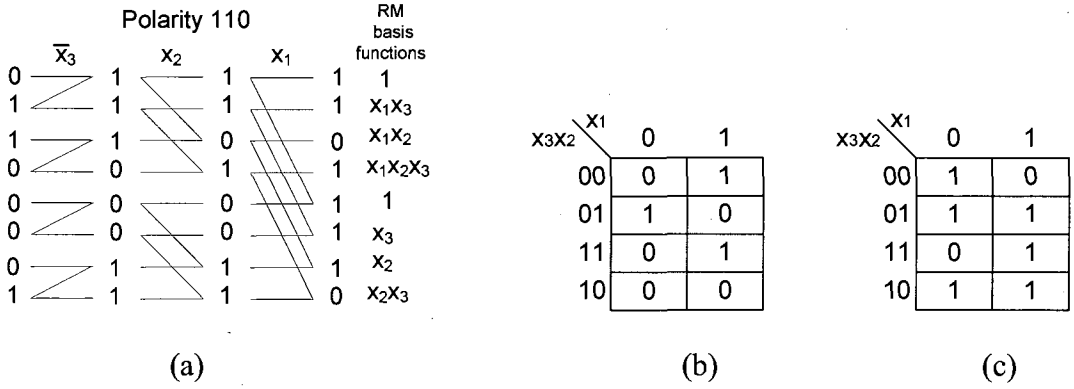


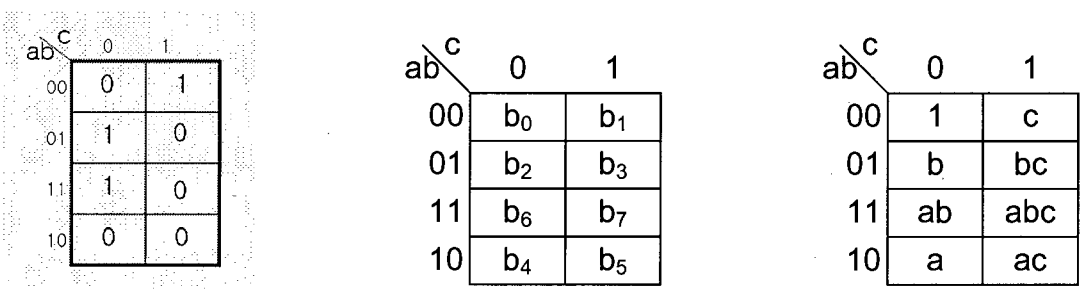
Figure 3.6.5: Conversion from minterms to FPRM with polarity 110. (a)Butterfly signal flow-graph for the polarity 110 of function F, (b) Karnaugh Map of the minterms and (c) Coefficient mapping in the Exor Map for polarity 110.

Figure 3.6.4 describes the Butterfly transformation equivalent to:

$$x_1 \bar{x}_3 \bar{x}_2 \oplus \bar{x}_3 x_2 \bar{x}_1 \oplus x_1 x_2 x_3$$
$$= x_1 (1 \oplus x_2 \oplus x_3 \oplus x_2 x_3 \oplus x_2 (1 \oplus x_1 \oplus x_3 \oplus x_1 x_3) \oplus x_1 x_2 x_3$$
$$= x_1 \oplus x_1 x_2 \oplus x_1 x_3 \oplus x_1 x_2 x_3 \oplus x_2 \oplus x_1 x_2 \oplus x_2 x_3$$
$$= x_1 \oplus x_2 \oplus x_1 x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3$$

As is described in detail in chapter 15 the butterfly diagrams may be created for any given polarity number for the FPRM expansion. Unfortunately, finding the "best" polarity number and its corresponding maximal number of zero-valued FPRM coefficients resulting in the "best" butterfly structure is very challenging. In this thesis, in chapter 15, we show how the use of quantum logic circuits can give an optimal polarity number more efficiently than any existing or even any possible algorithm for a standard computer.

3.6.1. Transformation from disjoint SOP to PPRM



(a) *(b)* *(c)*

Figure 3.6.1.1: PPRM transform for 3 variables (a) KMap to calculate the PPRM,

(b)SOP minterm Coefficients of Kmap cells for transformation of three variables calculated below in Figure 3.6.1.2, (c)the products of variables (base functions) for PPRM that correspond to the cells of the KMap and their respective coefficients from Figure 3.6.1.1b. The KMap from Figure 3.6.1.1c is related to EXOR Maps that will be introduced in chapter 8.

Based on coefficients and product terms from Figure 3.6.1.1b, c, we show the method to calculate the PPRM for the function from Figure 3.6.1.1(a). This is illustrated in Figure 3.6.1.2. The Figures explain several useful formalisms used to calculate the PPRM $c \oplus ac \oplus b \oplus abc$ of the initial function $\bar{a}\,\bar{b}\,c \oplus \bar{a}\,b\,\bar{c} \oplus ab\,\bar{c}$ from Figure 3.6.1.1a. Figure 3.6.1.3 shows the relation between the minterms and the b_i coefficients in a canonical SOP formula for minterms.

$b_0(a \oplus 1)(b \oplus 1)(c \oplus 1) \rightarrow b_0(1 \oplus a \oplus b \oplus c \oplus ab \oplus ac \oplus bc \oplus abc)$

$b_1(a \oplus 1)(b \oplus 1)c \rightarrow b_1(1 \oplus a \oplus b \oplus ab)c \rightarrow b_1(c \oplus ac \oplus bc \oplus abc)$

$b_2(a \oplus 1)b(c \oplus 1) \rightarrow b_2(1 \oplus a \oplus c \oplus ac)b \rightarrow b_2(b \oplus ab \oplus cb \oplus abc)$

$b_3(a \oplus 1)bc \rightarrow b_3(abc \oplus bc)$

$b_4a(b \oplus 1)(c \oplus 1) \rightarrow b_4(a \oplus ab \oplus ac \oplus abc)$

$b_5a(b \oplus 1)c \rightarrow b_5(abc \oplus ac)$

$b_6ab(c \oplus 1) \rightarrow b_6(ab \oplus abc)$

b_7abc

$bc \qquad bc(b_0 \oplus b_1 \oplus b_2 \oplus b_3)$

$ab \qquad ab(b_0 \oplus b_2 \oplus b_4 \oplus b_6)$

$abc \qquad abc(All)$

$\bar{a}\bar{b}c \oplus \bar{a}b\bar{c} \oplus ab\bar{c}$

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

$= (1 \oplus a \oplus b \oplus ab)c \oplus (1 \oplus a \oplus c \oplus ac)b \oplus ab(1 \oplus c)$

$= c \oplus ac \oplus bc \oplus abc \oplus b \oplus ab \oplus bc \oplus abc \oplus ab \oplus abc$

$= c \oplus ac \oplus b \oplus abc$

$PPRM = c_0 \cdot 1 \oplus c_1 \cdot c \oplus c_2 \cdot b \oplus c_3 \cdot bc \oplus c_4 \cdot a \oplus c_5 \cdot ac \oplus c_6 \cdot ab \oplus c_7 \cdot abc$

$c_i \in \{0,1\}$

Figure 3.6.1.2: Calculation of coefficients for the PPRM Circuit for function from Figure 3.6.1.1(a).

3.7. Conclusions to chapter 3.

Above we presented, based on published literature and research of the PSU quantum team, a unified description of basic ideas that we will use in the next chapters of the thesis. We gave also some examples discussing the importance of structure selection, transformation and analysis. Concluding, a successful method should take into account both the structure and the search algorithm for this structure. The next chapter will review some known structures and will introduce some new structures.

Based on this chapter I hope that I gave sufficient arguments for AND/EXOR logic as a base of algorithms for permutative quantum circuits synthesis and that I demonstrated also that the evolutionary programming approaches popular in the research literature on quantum synthesis are not sufficient. Our goal from now on will be to create efficient methods for AND/EXOR synthesis, for single-output and multiple-output function, that will allow for more efficient knowledge-based algorithms than those used so far and based on exhaustive group theory or evolutionary algorithms.

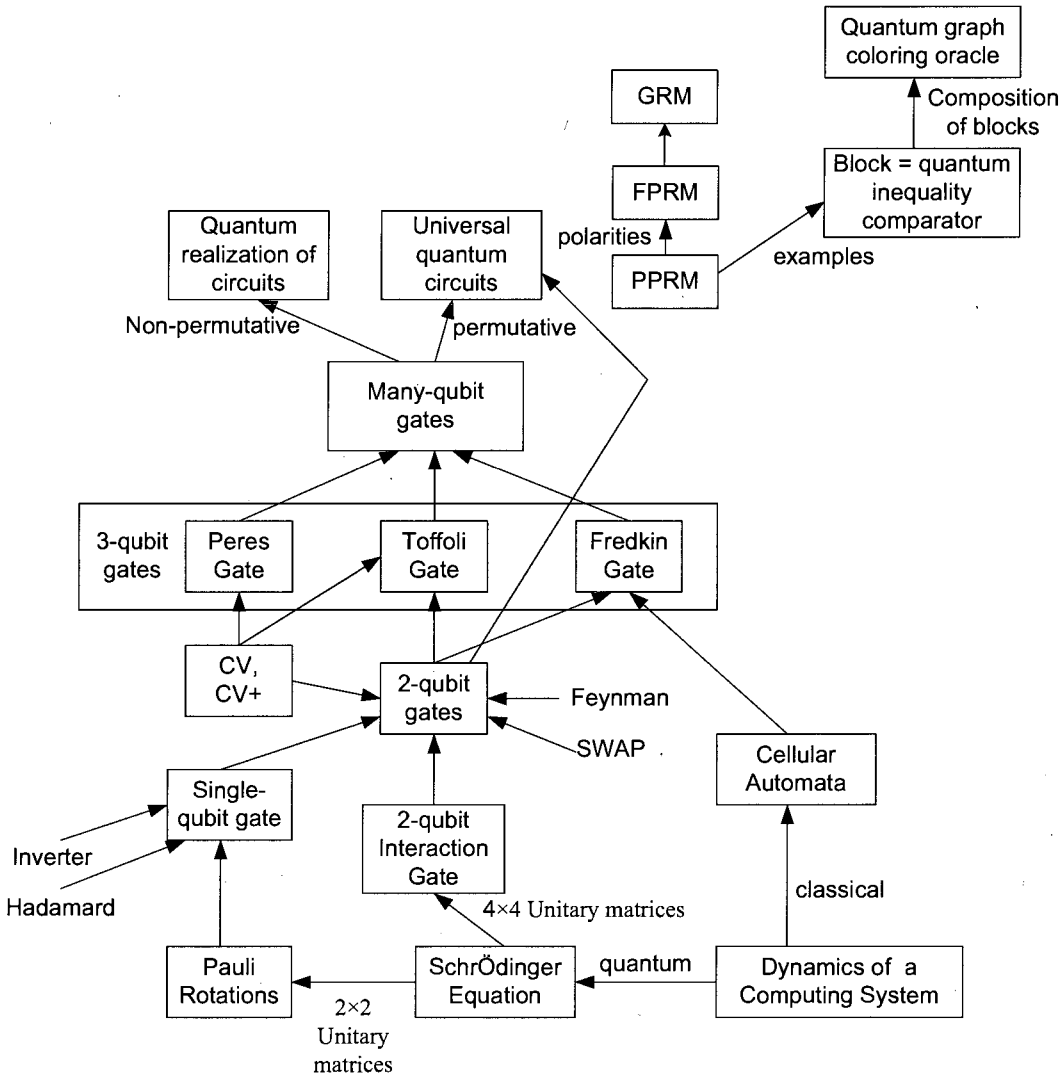


Figure 3.7.1: Diagrams of main concepts introduced in chapter 3.

Various decision diagrams used for switching functions can be uniformly regarded as graphical representations related to AND-EXOR expressions, derived by considering the switching functions as functions in the Galois Field, GF(2) [Stankovic97]. The diagram of the main concepts introduced in this chapter and their mutual relations is presented in Figure 3.7.1.

CHAPTER 4

Algebras, Expansions, Trees, Forms, Hypercubes and Quantum Arrays

As we have already shown, Reed-Muller (AND-EXOR) logic is fundamental to synthesize both "classical reversible" and "permutative quantum" circuits. It is still uncertain which forms of AND-EXOR logic are best as the first synthesis stage to create a reduced oracle. Moreover, it is not certain if Reed-Muller forms are the best choice for NMR-based quantum circuit design. A number of new algebraic families of complete operator sets and circuit structures, based on the Reed-Muller type logics, with particular interest in binary logic ESOP expressions (non-canonical) and their (canonical) subsets, as well as tree expansions, are introduced in Section 4.1.

4.1. Types of Logic

Several different algebraic systems of *"group based logics"* are developed in this dissertation. Boolean Logic, traditionally utilized in integrated circuit design, is restricted to binary input/output values and a few simple, basic gates, from which all circuits are built. With conventional CMOS technology and Boolean logic, a two-level AND-OR-structure-based implementation is most commonly utilized in practical designs. The algorithms for AND-OR minimization are based on the so-called "unate covering problem" (look chapter 6 for software realization and chapter 12 for quantum

oracle realization). The Unate covering can be also used as a part of the whole system for quantum circuit minimization. But, even in the case of contemporary VLSI CMOS technology the AND-EXOR implementation has been shown to be more economical, generally requiring fewer gates, fewer connections and smaller technology-related costs, much reduced (quantum) costs in quantum domain and high testability. In addition to quantum arrays, binary classical structures in standard notation will be also presented for completeness and as a link to classical circuits. This serves also as a link to classical CAD algorithms. Our main goal will be however to develop algorithms to synthesize quantum oracles and their partial logic blocks.

It is well known that the AND-EXOR form has been developed into a complete hierarchy of Reed-Muller (RM) Expansions (recalled in Section 4.1.1), using the Shannon [Shannon49], Positive Davio, and Negative Davio Expansions. This hierarchy is described with logic equation forms, trees, and decision diagrams [Sasao93e]. The AND-EXOR representations have interesting characteristics, allowing the representation of large functions and efficient representation of their properties.

Reed-Muller Logic Theory was also expanded with the introduction of the Generalized Kronecker Expansions to the Zhegalkin Hierarchy [Perkowski97a, Perkowski97c]. (Note that the Zhegalkin Kronecker Reed-Muller Form is obtained when a single expansion, from the set of all possible Zhegalkin linearly independent

forms is applied to every input variable. It was also recognized that *"the GRM expansion with functional coefficients is a special case of the LI expansion with functional coefficients"* [Perkowski97b]. Hence given a defining logic table, a method is presented for Zhegalkin Expansion, (either with or without functional coefficients), resulting in a valid GRM (Generalized Reed-Muller) with Linear Independence. With this understanding, the Reed-Muller Logic Hierarchy can be related to the Zhegalkin Hierarchy, as described by forms, trees, and decision diagrams [Perkowski97a, Perkowski97c]. While GRM is an AND/EXOR form the general LI forms are not. The Zhegalkin Hierarchy is a subset of the Linearly Independent Logic Hierarchy [Perkowski97]. It includes the Reed-Muller Hierarchy and all other AND/EXOR forms, both those presently known and those to be found in future. (Note that the Linearly Independent Hierarchy is not restricted to circuits built from AND and EXOR gates in the binary case). The Zhegalkin Hierarchy is named in honor of the Russian scientist who in 1927 discovered the forms [Zhegalkin29] now attributed to Reed and Muller's research published in 1954 [Reed54]. These forms were the starting point to the whole research area of "EXOR logic" that influenced heavily the classical and quantum circuit design.

I prefer the AND-EXOR logic structures because of their good match to quantum technology. Even for the case of AND-EXOR logic, the Sum-of-Products (SOP) expansion is the popularly used general form for specifying a given Boolean function. SOP or POS are used by necessity in many oracles. These two expression types we

call regular because one can clearly distinguish two planes in them – the AND plane and the OR plane which are both structures of regularly placed gates and planes are abutting. This is important for possible hardware layout purposes. The concept of regularity is important in classical logic design with many applications in PLAs, PLDs, and FPGAs. In addition, the SOP (and especially Disjoint SOP or DSOP) is a good starting point from which to develop the logic family hierarchy and problem specifications. Methodically, starting from expansions, then trees, followed by decision diagrams, and finally two-level forms, the complete new linearly independent logical family hierarchies will be developed in chapters 8, 9 and 10. However using only AND and OR gates one cannot build an arbitrary function. Negations are also needed. Thus AND/EXOR logic is more powerful in the sense that AND and EXOR gates together with the constant "1" form a universal logic system (the NOT gate is not needed). The families based on AND/EXOR logic are more extended and more interesting than AND/OR circuit. Most importantly, they are superior in quantum circuit design.

The main observation is that in quantum logic many gates used as operators are the algebraic structures called groups. We will call all of them the group-based logics. (Note [Stewart89]):

Definition 4.1.1: A group (G,*) is a non-empty set G and a binary operator (*) on G, such that the following conditions hold:

Closure: For all a, b $\in$ G, a*b $\in$ G.

Associativity: For all a, b, and c in G, a * (b * c) = (a*b) * c.

Identity: There exists an identity element "e" $\in$ G, such that a * e = e * a = a, for all a $\in$ G.

Inverse: For each a $\in$ G, there exists an inverse $a^{-1} \in$ G, such that $a * a^{-1} = a^{-1} * a = e$.

The symbol * here is the so-called group operator and should not be confused with multiplication. It is rather like an EXOR.

Definition 4.1.2: A non-empty set F is called a **field** when the two-argument operations "+" and "*" are defined on F and the following properties hold. They are called a sum and a product, respectively (addition and multiplication).

Closure:

For all a, b $\in$ F, a + b $\in$ F and a*b $\in$ F.

Associative:

For all a, b, and c $\in$ F, a + (b + c) = (a + b) + c and a * (b * c) = (a * b) * c.

Commutative:

For all a, b $\in$ F, a + b = b + a and a * b = b * a.

Distributive:

For all a, b, and c $\in$ F, a * (b + c) = a * b + a * c and (a + b) * c = a * c + b * c.

Identity:

For all $a \in F$, $a + 0 = a$, (since 0 is the additive identity) and $a * 1 = a$, (since 1 is the multiplicative identity).

Inverse:

For each $a \in F$, there exist inverses $-a \in F$ and $a^{-1} \in F$, such that $a + (-a) = 0$, (since $-a$ is the additive inverse) and $a * a^{-1} = 1$, (since a^{-1} is the multiplicative inverse).

In brief, a simple definition is that Galois Field (GF) is algebra with finite set of elements, and two operations, the Galois addition and Galois multiplication. There are certain operators expressed in Tables in Figure 4.1 .1. Figure 4.1 .1a shows GF(3) addition of elements, which shows that if we have a and b inputs in the table, then output will be defined inside the table. And those operations have to satisfy certain Axioms (Definition 4.1 .1 and Definition 4.1 .2). The Galois Field addition for 3-valued (ternary) variables (GF(3) add), Figure 4.1 .1a which is the same as the modulo addition. In the Figure 4.1 .1, we have arguments which are always the values of 0, 1, 2 -that is a ternary system. As we see, the table in Figure 4.1.1a is a Latin Square, every row and every column is different and uses all elements 0, 1 and 2. In Figure 4.1.1b is Galois Multiplication for 3 valued variables (GF(3) *) which is the same as the modulo-3 multiplication. Again, we find Latin Square in Galois Field multiplication.

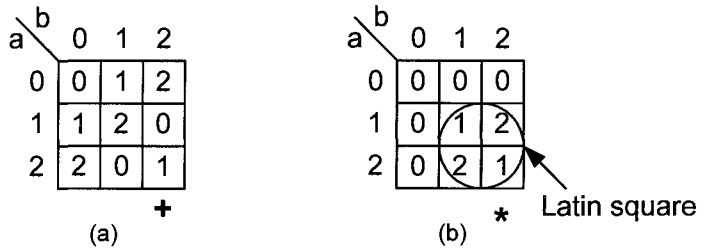


*Figure 4.1.1: (a) GF(3) Logic operators Table of Galois Field addition for 3-valued variables (GF(3)add). It is Latin Square (b) Table of Galois Field multiplication for 3-valued variables (GF(3) *).*

Here in Figure 4.1.2a we showed the Galois Field addition for 4-valued variables. Galois Addition here we have to calculate as the Boolean vector exoring. For example, if we have vectors (0,1) and (0,1), then we need to do exoring bit-by-bit which produces a 0 for adding the 1 and 1 arguments. As like Galois Field Multiplication for 4 variables in Figure 4.1.2b, also creates a Latin Square – non-zero values. The multiplication operation is defined to satisfy all GF axioms together with the addition operation.

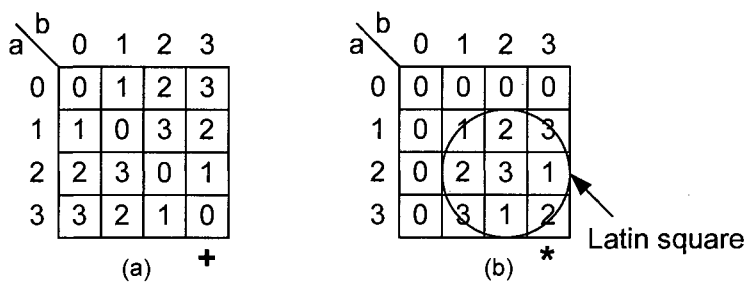


Figure 4.1.2: GF(4) Logic operators. (a) Table of Galois Field addition for 4 variables (GF(4)add). It is a Latin Square. (b) Table of Galois Field multiplication for 4-valued variables (GF(3)). It is not a Latin Square but it includes a Latin Square.*

We found that all binary logic synthesis methods from this thesis can be extended to arbitrary Galois Fields but this is not a subject of this thesis.

4.2. Binary Reed-Muller Logic.

The Reed-Muller Hierarchy is well known from the literature (called also Green hierarchy or Green-Sasao hierarchy) [Sasao97f]. Such circuits have desirable properties including: 1) requiring fewer product terms for many classes of functions [Dill97] and 2) desirable testing properties [Biamonte04, Biamonte05, Biamonte05b, Perkowski07, Pierce05]. Here only a limited review of the definitions useful for this thesis and the hierarchy of forms will be presented. There are several different classifications of AND-EXOR expressions [Sasao90a]. All Reed-Muller forms are canonical, as there exists only one function expression (called the form) for the given polarity of variables. The general expression for several Reed-Muller forms (including GRM, FPRM and PPRM) is given as follows:

$$f(x_1,x_2, \ldots, x_n) = a_0 \oplus a_1X_1 \oplus a_2X_2 \oplus \ldots \oplus a_nX_n \oplus a_{12}X_1X_2 \oplus a_{13}X_1X_3 \oplus \ldots \oplus a_{n-1,n}X_{n-1}X_n \oplus a_{12\ldots n}X_1X_2X_3\ldots X_n$$

where, a's are binary constants and X_i are literals, $X_i = x_i$ or x_i'.

Function expansions are the basis for the derivation of the hierarchy of logic families. The Shannon, Positive Davio, and Negative Davio are well known [Sasao1] and we

discussed them in Chapter 3 as the fundament of this thesis. Here we repeat the formulas for reader's convenience. Given an arbitrary logic function, $f(x_1, x_2, x_3,..., x_n)$, where the cofactors are $f_0 = f(0, x_2, x_3,..., x_n)$, and $f_1=f(1, x_2, x_3,..., x_n)$, and Boolean difference is $f_2 = f_0 \oplus f_1$, these expansions are recalled as follows.

Positive Davio Expansion: $f = 1*f_0 \oplus x_1*f_2 = f_0 \oplus x_1 f_2$

Negative Davio Expansion: $f = 1*f_1 \oplus \bar{x}_1*f_2 = f_1 \oplus \bar{x}_1 f_2$

Shannon Expansion: $f = \bar{x}_1 f_0 \oplus x_1 f_1$

The following definitions describe each of the specific forms within the Reed-Muller Hierarchy. The above expansions can be generalized to expansions with respect to groups of binary variables and to expansions for Galois Fields.

The most restrictive form in the binary classification is the Positive Polarity Reed-Muller (PPRM) expression form. This type of equation is canonical, with less than or equal to 2^n products of only positive literals. It is more formally defined as follows.

Definition 4.2.1: The Positive Polarity Reed-Muller Form (PPRM) is a Reed-Muller expression in which all literals of variables are expressed with positive polarities. The expansion tree [Sasao89] is given in Figure 4.2.1.

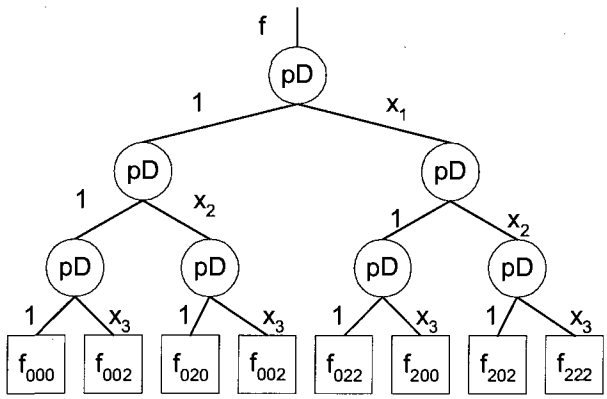


Figure 4.2.1: Expansion tree for the Positive Davio Expansions.

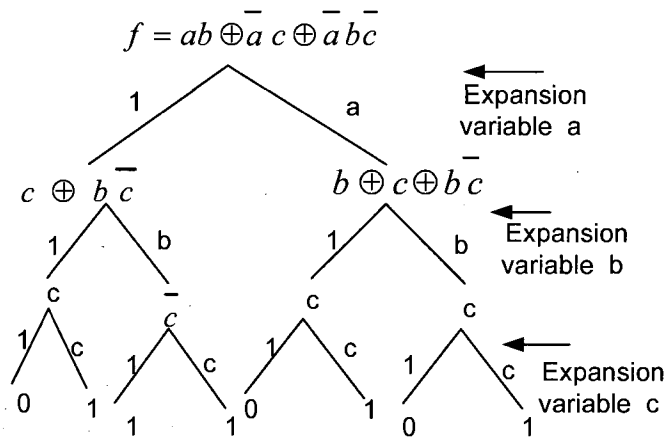


Figure 4.2.2: An Example of using positive Davio expansions to calculate the expansion tree for order of variables a, b, c.

It can be verified by exoring all branches in Figure 4.2.2 leading to "1", thus

$$\hat{f} = c \oplus b \oplus bc \oplus ac \oplus abc \;=\; c\bar{a} \oplus b(1 \oplus c \oplus ac) = c\bar{a} \oplus b(1 \oplus c\bar{a}) = c\bar{a} \oplus b(a \oplus \bar{a}\bar{c}) = c\bar{a} \oplus ab \oplus \bar{a}b\bar{c} = f$$

combining isomorphic nodes the tree is converted to a decision diagram DD [Sasao93e].

Example 4.2.1: Given is $f = ab \oplus \overline{a}\, c \oplus \overline{a}\, b\, \overline{c}$. The tree expanded using only positive Davio expansions is shown in Figure 4.2.2. The multilevel circuit corresponding to this tree is given in Figure 4.2.3. After flattening the tree, one obtains $f = c \oplus b \oplus bc \oplus ac \oplus abc$.

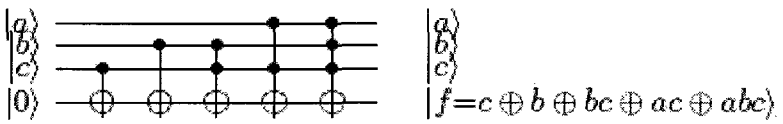
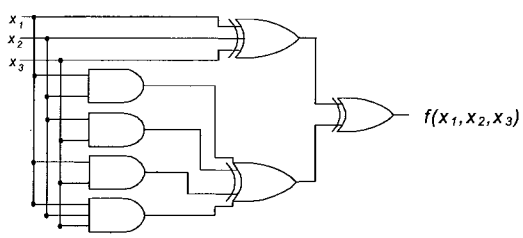


Figure 4.2.3: The PPRM form realized as a quantum array using Feynman and Toffoli gates for the function $f = c \oplus b \oplus bc \oplus ac \oplus abc$ *from Figure 4.2.2.*

Example 4.2.2: An example PPRM is given:

$f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2x_3$

The classical diagram for this function is drawn in Figure 4.2.4a and its corresponding quantum array is presented in Figure 4.2.4b.



(a)

(b)

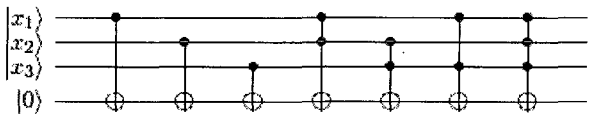


Figure 4.2.4: Circuits for Example 4.2.2. (a) The PPRM circuit as a classical diagram, (b) PPRM form realized as a quantum array using Feynman gates and unrestricted Toffoli gates.

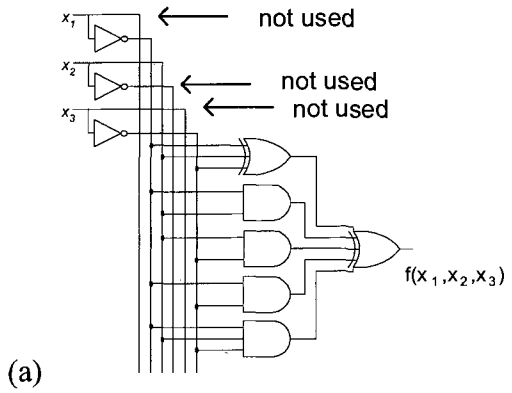
Another form in the classification for AND-EXOR equations is the Fixed Polarity Reed-Muller (FPRM) form which includes the PPRM types of expressions (forms) as a special case. The FPRM expression can contain either positive or negative literals for each variable, but not a mixture of product terms with various polarities of the same variable. This FPRM expression is canonical, having unique coefficients, and usually requires fewer terms than the PPRM form, but can never have more terms. Because of a low cost of inverters (realized in quantum as single Pauli X rotations) in all quantum technologies the optimal FPRM is always not worse (in most cases better) than the PPRM form, at least it is so far the single output functions.

Definition 4.2.2: The Fixed Polarity Reed-Muller Form (FPRM) is defined as a Reed-Muller expression in which all the literals of a variable can be either positive or negative, but cannot exist in both polarities.

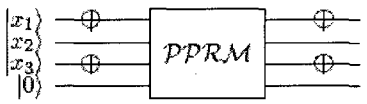
Example 4.2.3: An example of a FPRM is given.

$$f(x_1, x_2, x_3) = \bar{x}_1 \oplus x_2 \oplus \bar{x}_3 \oplus \bar{x}_1 x_2 \oplus x_2 \bar{x}_3 \oplus \bar{x}_1 \bar{x}_3 \oplus \bar{x}_1 x_2 \bar{x}_3$$

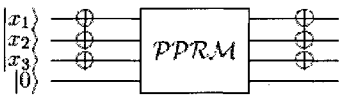
The variables x_1 and x_3 have negative polarities, while x_2 has a positive polarity throughout the expression (Figure 4.2.5). Two Quantum arrays for two different polarities are shown in Figure 4.2.5c.



(a)



(b)



(c)

Figure 4.2.5: FPRM forms and their diagrams: (a) classical schematics of FPRM from Example 4.2.3, (b) The realization of the function from Example 4.2.3 as a schematic quantum array; the PPRM in the box realizes function $\hat{x}_1 \oplus \hat{x}_2 \oplus \hat{x}_3 \oplus \hat{x}_1\hat{x}_2 \oplus \hat{x}_2\hat{x}_3 \oplus \hat{x}_1\hat{x}_3 \oplus \hat{x}_1\hat{x}_2\hat{x}_3$, *here* $\hat{x}_1 = \bar{x}_1$, $\hat{x}_2 = x_2$, $\hat{x}_3 = x_3$. *(c) the circuit for negative polarity FPRM form.*

Given an arbitrary Reed-Muller Expression, to formulate the FPRM form, either Positive or Negative Davio Expansions are substituted for each variable x_i (i = 1, 2, 3, ..., n) in subsequent expansions.

Example 4.2.4: The FPRM form is derived for the two-level AND-EXOR expression [Sasao81] given below:

$f(x_1, x_2, x_3, x_4) = x_1 x_2 x_3 x_4 \oplus \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$

Variables x_1 and x_2 should have positive polarity and variables x_3 and x_4 should have negative polarity. The circuit should be minimized not as an oracle.

Substitutions for variables, with identity expressions, are made as follows:

$\bar{x}_1 = x_1 \oplus 1$

$\bar{x}_2 = x_2 \oplus 1$

$x_3 = \bar{x}_3 \oplus 1$

$x_4 = \bar{x}_4 \oplus 1$

Thus,

$f(x_1, x_2, x_3, x_4) = x_1 x_2 x_3 x_4 \oplus \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$

$f(x_1,x_2,x_3,x_4) = x_1 x_2(\bar{x}_3 \oplus 1)*(\bar{x}_4 \oplus 1) \oplus (x_1 \oplus 1)(x_2 \oplus 1)\ \bar{x}_3\ \bar{x}_4$

$f(x_1, x_2, x_3, x_4) = x_1 x_2 (1 \oplus \bar{x}_3 \oplus \bar{x}_4 \oplus \bar{x}_3\ \bar{x}_4) \oplus (1 \oplus x_1 \oplus x_2 \oplus x_1 x_2)\ \bar{x}_3\ \bar{x}_4$

$f(x_1, x_2, x_3, x_4) = x_1x_2 \oplus x_1x_2\bar{x}_3 \oplus x_1x_2\bar{x}_4 \oplus \bar{x}_3\ \bar{x}_4 \oplus x_1\bar{x}_3\ \bar{x}_4 \oplus x_2\bar{x}_3\ \bar{x}_4$

This FPRM form with negative polarities of variables x_3 and x_4 is realized as a quantum array in Figure 4.2.6.

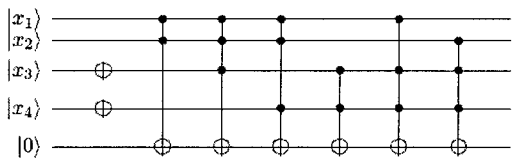


Figure 4.2.6: The quantum array of the FPRM form derived in Example 4.2.4 (not an oracle).

In Figure 4.2.6 variables x_3 and x_4 are not restored and remain negated. Observe that outputs x_3 and x_4 have inverted values. This is not acceptable if the whole circuit is used as a single oracle in which case every input variable must be repeated in unchanged form on the oracle's output to be measured together. It is however acceptable in a circuit being a sub-block of an oracle.

Definition 4.2.3: The Kronecker Reed-Muller Form (KRO), KRM is derived by the application of the Shannon, Positive Davio, or Negative Davio Expansions, with the restriction that only one type of expansion can be applied per level in the ordered tree (one variable per level).

Example 4.2.5:

An example of a general KRO expansion Tree [Sasao86] is shown in Figure 4.2.7. A KRO tree for function $f = ab + (c \oplus \bar{a}\, b)$ is given in Figure 4.2.8a. The flattened KRO form obtained from this tree is given in Figure 4.2.8b.

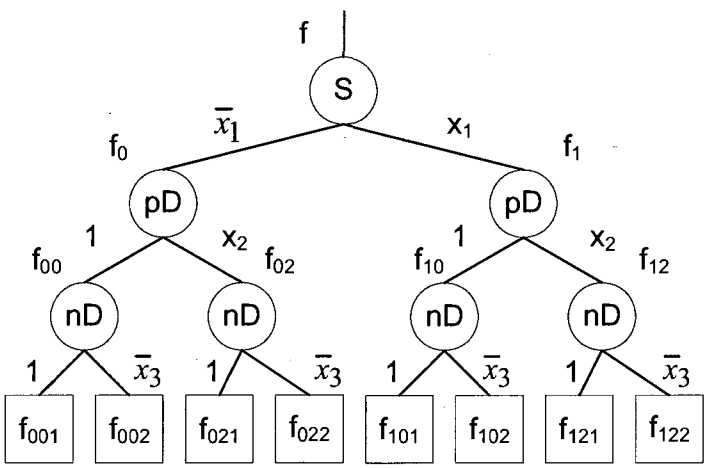
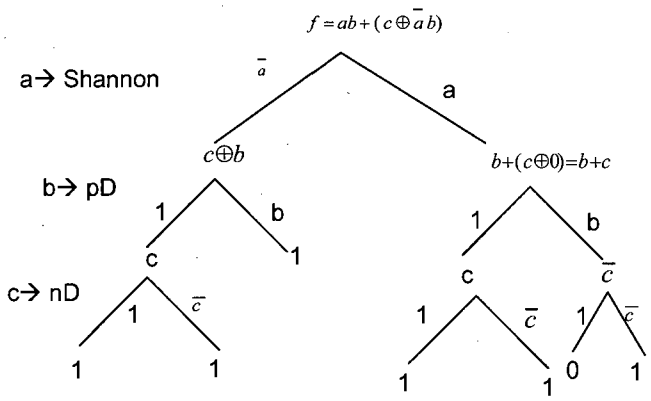


Figure 4.2.7: A general form of a KRO Expansion Tree. In this case Shannon Expansion is applied to variable x_1, positive Davio Expansion is applied to variable x_2 and Negative Davio Expansion is applied to variable x_3.



(a)

(b) $\bar{a} \oplus \bar{a}\,\bar{c} \oplus \bar{a} b \oplus a \oplus a\bar{c} \oplus ab\bar{c}$

(c)

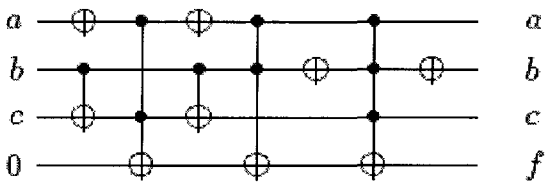


Figure 4.2.8: An oracle for a KRO expansion.(a) A practical Example of KRO Expansion Tree using Shannon Expansion, Positive Davio and Negative Davio Expansion in order of variables a, b,c, (b) the flat KRO equation form obtained from this tree, (c)The quantum array obtained directly from this tree: $\bar{a}(b \oplus c) \oplus a(b+c) = \bar{a}(b \oplus c) \oplus ab \oplus a\bar{b}c$ *. Mirror added to restore input c. Observe that all inputs are restored as this is an oracle.*

An algebraic expression can be constructed from an expansion tree by combining the coefficients for each expansion, with Boolean multiplication along each branch to the leaf, and then combining the branches (product terms) with the exclusive-or operation.

Example 4.2.7: Write the algebraic expression for the example KRO Expansion tree shown in Figure 4.2.7.

$f(x_1, x_2, x_3) = \bar{x}_1 * 1 * 1 * f_{001} \oplus \bar{x}_1 * 1 * \bar{x}_3 * f_{002} \oplus \bar{x}_1 * x_2 * 1 * f_{021} \oplus \bar{x}_1 * x_2 * \bar{x}_{13} * f_{022} \oplus x_1 * 1 * 1 * f_{101} \oplus x_1 * 1 * \bar{x}_3 * f_{102} \oplus x_1 * x_2 * 1 * f_{121} \oplus x_1 * x_2 * \bar{x}_3 * f_{122}$

The quantum array can be drawn from the above expression after substituting the values of f_{ijk} with binary constants 0, 1.

Definition 4.2.4: The Pseudo Reed-Muller Form (PSDRM) is obtained by applying the Positive Davio and Negative Davio Expansions, but with different expansions for each sub-function, as desired. This means that observing the expansion tree for the PSDRM, either the Positive or Negative Davio expansions are applied at will, regardless of the node or level of the tree, and different types of expansions may be used for the same variable [Sasao94].

Example 4.2.8:

An example of a PSDRM Expansion Tree is given in Figure 4.2.9. In this tree, each level consists of both Positive and Negative Davio Expansions.

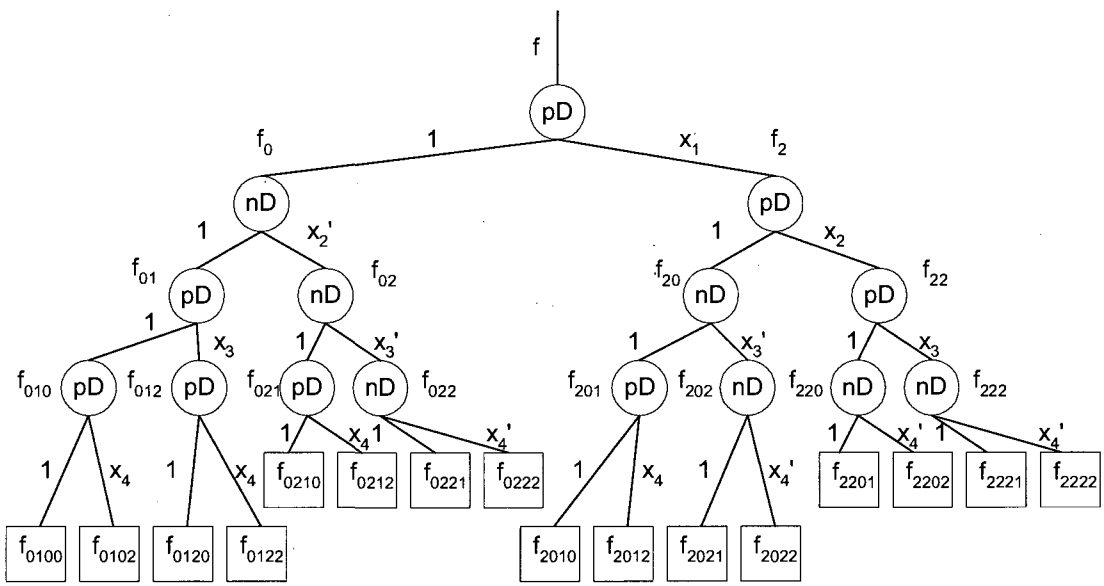


Figure 4.2.9: An Example of a PSDRM tree.

Example 4.2.9: For the function $f(x_1, x_2, x_3, x_4) = x_1 x_2 x_3 x_4 \oplus \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$, the particular PSDRM form described by the tree in Figure 4.2.9 can be applied to produce the expansion tree shown in Figure 4.2.10 below [Sasao95f]. The Expression obtained by flattening the tree from Figure 4.2.9 is the following: $\overline{x_2}\,\overline{x_3} \oplus x_1 \overline{x_3}\,\overline{x_4} \oplus x_1 x_2 \overline{x_4} \oplus x_1 x_2 x_3$.

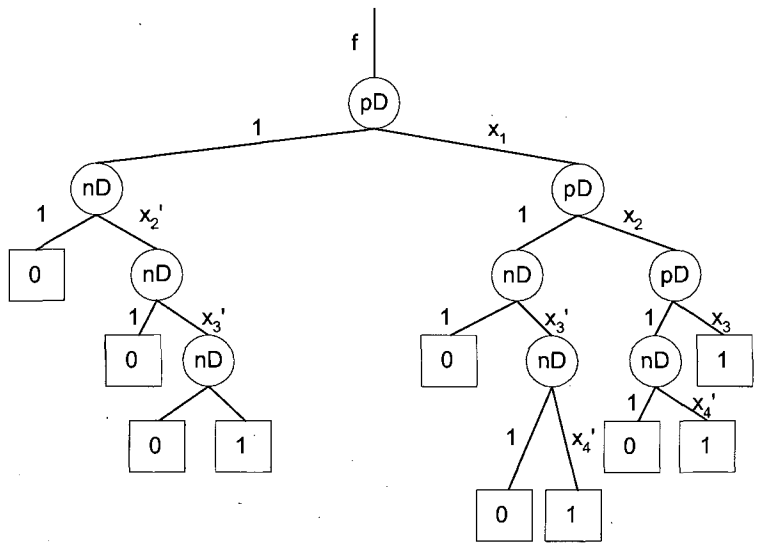


Figure 4.2.10: The PSDRM tree for $f(x_1,x_2,x_3,x_4) = x_1 x_2 x_3 x_3 \oplus \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$ *in which has the flattened PSDRM form of* $\overline{x_2}\,\overline{x_3} \oplus x_1 \overline{x_3}\,\overline{x_4} \oplus x_1 x_2 \overline{x_4} \oplus x_1 x_2 x_3$.

The quantum array corresponding to the flattened expression is shown in Figure 4.2.11. It is an oracle.

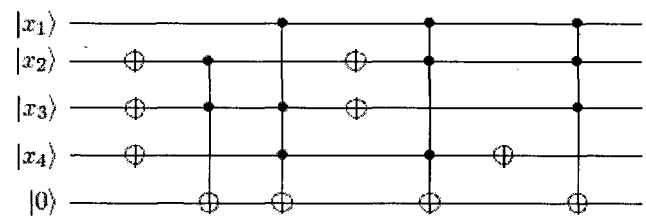


Figure 4.2.11: The quantum array for the flattened PSDRM form from Example 4.2.9.

Definition 4.2.5: The Pseudo-Kronecker Reed-Muller Form (PSDKRO) is derived by the application of any subset of Shannon, Positive Davio, and Negative Davio Expansions in an expansion tree level, but the tree is ordered (with the same order of variables for each branch). Herein, the tree is defined and introduced as desired.

Definition 4.2.6: The Free Kronecker Reed-Muller Form (FKRM) is not canonical and is derived by the application of the Shannon, Positive Davio, and Negative Davio Expansions, without restrictions and with no ordering of variables.

As may be recalled, while more restrictive than the Exclusive-Or Sum-of-Products (ESOP) expression, the GRM form incorporates the Fixed-Polarity Reed-Muller (FPRM) and the Positive Polarity Reed-Muller (PPRM) Forms as its special cases. Of course, the GRM is canonical for each of its polarities.

Definition 4.2.7: The Generalized Reed-Muller Form (GRM) is a general, canonical expression of the Exclusive-Or-Sum-of-Products type, in which for every subset of input variables, there exists at most one term with any arbitrary polarities of all variables. Thus for an n-variable function there are $n2^{n-1}$ literals and $2^{n2\wedge(n-1)}$ polarities. The GRM expansion of an n-variable function is shown as:

$$f(x_1, x_2, \ldots, x_n) = a_0 \oplus a_1 X_1 \oplus \ldots \oplus a_n X_n \oplus a_{12} X_1 X_2 \oplus a_{13} X_1 X_3 \oplus \ldots \oplus a_{n(n-1)} X_n X_{n-1} \oplus \ldots \oplus a_{12\ldots n} X_1 X_2 \ldots X_n$$

Where,

X can be expressed either as a positive literal x_i or a negative literal x_i'

$X = (X \oplus \delta)$

$\delta = 0/1$ for positive/negative polarity

a_i = coefficient of X_i, and can be 0 or 1

Example 4.2.10: An example of a GRM is given as $f(x_1, x_2, x_3) = \bar{x}_1 \oplus x_1 x_2 \oplus \bar{x}_3 \oplus x_1 x_2 x_3$.

The quantum array corresponding to this expression is shown in Figure 4.2.12. As we see, the quantum array for GRM can be represented as a concatenation of arrays for certain FPRMs. This observation can be used to create efficient iterative algorithms that find the best FPRMs for the remainder sub-functions of the initial function and concatenate them.

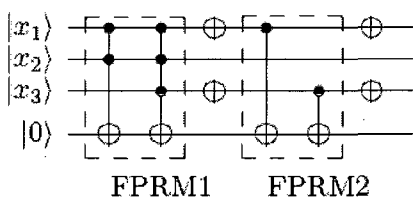


Figure 4.2.12: The quantum array oracle for the Generalized Reed Muller form of the function from example 4.2.10.

Finally, the most general classification of AND-EXOR equations, including the PPRMs, FPRMs, and GRMs classifications, is the Exclusive-Or-Sum-of-Products

(ESOP) expression. Because the ESOP form is not a canonical expression, there are no restrictions on the terms in its expression. Defined loosely, it is an expression simply consisting of arbitrary product terms with arbitrary literals, negated or not, combined with the EXOR operations.

Definition 4.2.8: The Exclusive-Or-Sum-of-Products expression (ESOP) is a non-canonical form in which arbitrary product terms are combined using EXOR logic gates [Sasao94].

The relations between the classical, binary Reed-Muller expressions are shown in Figure 4.2.13. (This does not include Zhegalkin forms.) The forms illustrated in this diagram have the following inclusion relations: (1) PPRM $\subset$ FPRM, (2) FPRM $\subset$ PSDRM, (3) FPRM $\subset$ KRO, (4) KRO $\subset$ PSDKRO, (5) PSDRM $\subset$ PSDKRO, (6) PSDKRO $\subset$ FKRM, and (7) PSDRM $\subset$ GRM. (Note that the GRM Form, while still canonical, usually requires nearly as few terms as the ESOP Form.) The families of expressions shown in dashed lines, FKRM and ESOP, are not canonical.

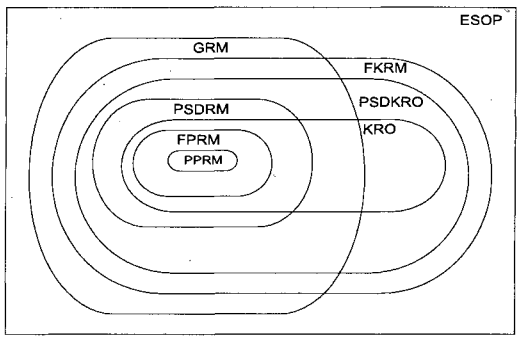


Figure 4.2.13: The hierarchy of the popularly known canonical forms and AND/EXOR expressions. This is also called Green hierarchy or Green-Sasao hierarchy.

A subset of the Reed-Muller Hierarchy is given with corresponding expansions, trees, decision diagrams, and forms in Figure 4.2.14.

Expansion	Tree	Diagram	Form
Shannon Expansion (S)	Shannon Tree	Binary Decision Diagram (BDD)	Sum-of- Product Canonical Form
Positive Davio Expansion (pD)	Positive Davio Tree	Functional Decision Diagram (FDD)	Positive Polarity Reed- Muller Form (PPRM)
Shannon, Positive and Negative Davio (S, pD, nD) (But only one type of expan-sion per level)	Kronecker Tree	Kronecker Decision Diagram (KDD)	Kronecker Reed- Muller Form (KRM)
Shannon, Positive, and Negative Davio (S, pD, nD) (But any subset in every level)	Pseudo- Kronecker Tree	Pseudo- Kronecker Decision Diagram (PKDD)	Pseudo- Kronecker Reed- Muller Form (PKRM)
Shannon, Positive, and Negative Davio (S, pD, nD) (No order of Variables)	Free Kronecker Tree	Free Kronecker Decision Diagram (FKDD)	Free Kronecker Reed- Muller Form (FKRM)

Figure 4.2.14: Classical, Green-Sasao hierarchy of Reed-Muller binary trees, diagrams and expansions

As previous research has shown, the relations of these forms in the Reed-Muller Hierarchy correspond directly to and have counterparts in the Zhegalkin [Perkowski97b] and Galois Field Hierarchies [Dill97, Dill97a, Dill97b, Dill97c]. The Zhegalkin Hierarchy includes all (including Reed-Muller) AND/EXOR canonical forms (also trees, decision diagrams, and expansions), both presently known and those that may be developed in the future, which are created by linearly independent, AND/EXOR expansions (i.e. Shannon, Positive and Negative Davio, GRMs, etc.). For each of structures in Figure 4.2.14 a corresponding quantum array can be created. The examples given so far should convince the reader that the above statement is true.

4.3. Representation of AND/EXOR Logic – The Polarity Maps

As we have seen, the concept of polarity is very useful for synthesis of AND/EXOR forms. The problem is, how can we systematically find all 2^n FPRMS for a given function F. There are several methods known from the literature to achieve this task, for instance using butterfly diagrams or matrix multiplication, but faithful to our graphical approach from this thesis, I will show the graphical methods based on the Exor Maps introduced originally by Tran [Tran89]. My method presented here is simpler and more intuitive. All the FPRM forms can be generated by changing the polarities of the variable in the PPRM Exor Map, one variable at a time.

For each of 2^n maps we can use EXOR rules to factorize or resynthesize the expression from the map in the best possible way.

All $2^n = 2^3 = 8$ FPRMs for 3 variables are shown in Figure 4.3.1 below:

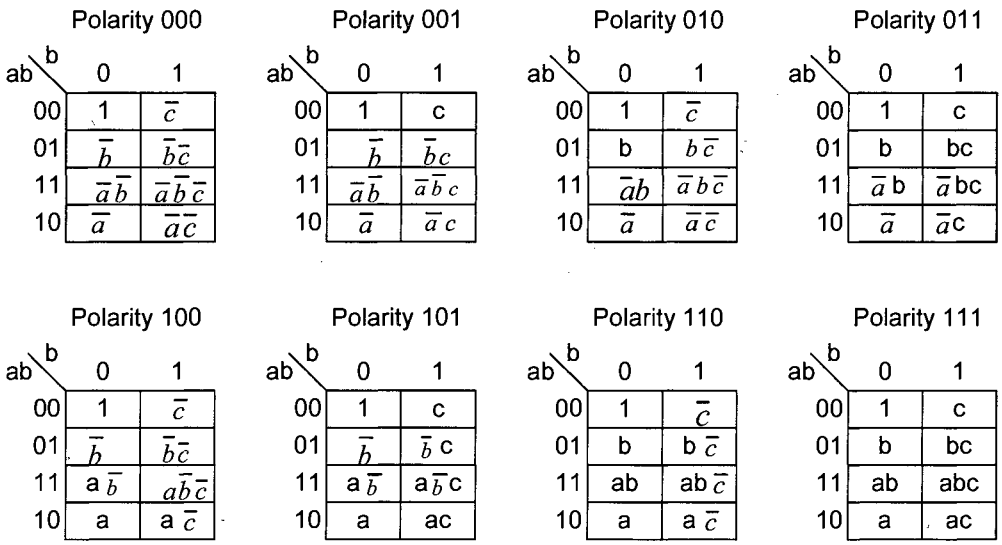


Figure 4.3.1: The Exor Maps for all FPRM polarities for functions of three variables.

Observe that the sets of variables are the same in all corresponding cells of these maps, but the variables are negated or not in them, according to the respective (consistent or "fixed") polarity. Now our question is how to transform the same function to find its representation in a new polarity, knowing the representation of this function in the previous polarity.

The transformation rules for variables in product terms are the following:

Transformation rule

X→0 (for polarity=1)

X→1 (for polarity=0)

1→1 (for polarity=1)

1→0 (for polarity=0)

0→X (for polarity 0 or 1)

4.3.1. Transforming a KMap to an EXOR Map.

If f is the Exor Map function and x is the KMap function and p is the polarity of the Exor Map then Exor Map function can be generated from the KMap function using the following rules

Rule-1: $F_i = X$ if $x_i=0$

Rule-2: $F_i = p_i$ if $x_i=1$

Rule-3: $F_i = p_i'$ if $x_i=X$

Note: Reduction of the Kmap may be extended using the ESOP minimization, before applying the transformation rules.

Proof for Rule-1

In the Exor Map for any polarity the position 000…0 in it is always XXX...X

Whereas in Kmap it is always 000…0

Therefore the transformation rule is: 0→ X

Proof for Rule-2

In the exor map for any polarity the position 111…1 it is always the same as the polarity,

Whereas in Kmap it is always 111…1

So the transformation rule is

1→1 (for polarity=1)

1→0 (for polarity=0)

Therefore it is the same as the polarity.

Proof for Rule-3

Let us consider first an example.

Example 4.4.1 :

Given is the Exor Map in polarity $\bar{a}\,b\,c$ from Figure 4.3.2.

ab \ b	0	1
00	1	c
01	b	bc
11	$\bar{a}$ b	$\bar{a}$ bc
10	$\bar{a}$	$\bar{a}$ c

Figure 4.3.2: Exor Map for polarity $\bar{a}\,b\,c$.

From KMap

F=b

=X1X

Using transformation rules

Polarity of the Exor map: 011

X1X→110

From the Exor Map we obtain

F=b⊕ bc ⊕ a'b ⊕ a'bc

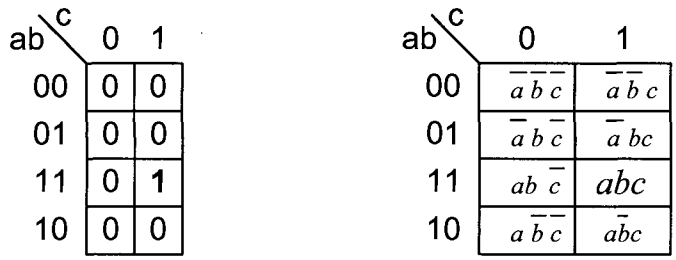
=abc'

=110

From the above example it can be inferred that symbol X from the KMap are converted to 0 if the polarity of the variable is one and to 1 if the polarity of the variable is 0, which is nothing but the negation of the polarity values. End of proof.

Example 4.3.2:

Here we give a complete example of this kind of transformations for a function from a PPRM as in Figure 3.3.3a to another FPRMs of another polarities.



(a) (b)

Figure 4.3.3: Representations of Example 4.3.2. (a) Positive polarity Exor Map representing function f = abc. (b) Symbolic representation of standard minterms in a standard Kmap.

We explain in brief how we transform from polarity 111 of function f = abc. Figure 4.3.3(b) gives a symbolic representation of minterms as products of literals in a KMap for Figure 4.3.3a assumes polarity 111 (a = 1, b = 1, c = 1). Figure 4.3.3(a) is the Exor Map that represents the function as a set of its minterms. Each cell is a positive polarity product. Here in Figure 4.3.4 we go systematically in Gray code through the polarities of 111, 101 and 100 and using the transformation rules we find the Exor Maps of f for their corresponding polarities.

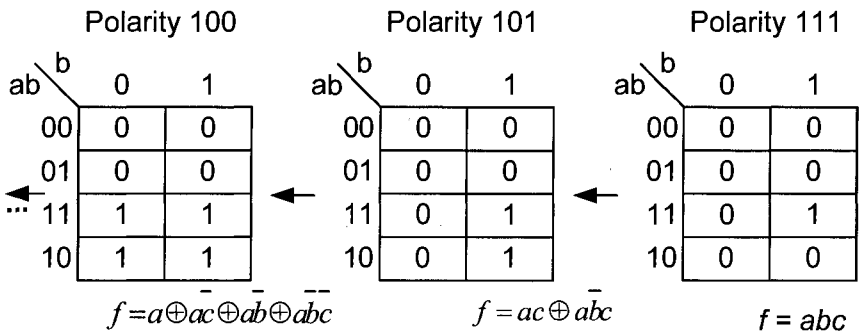


Figure 4.3.4: Visual Transformation of FPRM from Example 3.4.2 in different polarities, (111), (101) and (100). Similarly all FPRMs can be generated changing a polarity of one variable at a time.

4. 4. Gray-code based systematic generation of all FPRM forms.

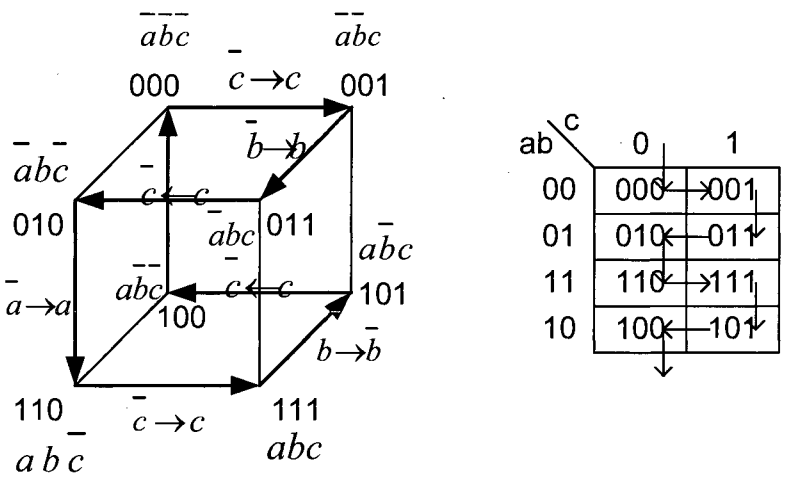


Figure 4.4.1: Hamming distance 1 path (HD1 path) through all nodes in a 3-dimensional hypercube.

In this section we denote polarities as binary numbers and not decimal numbers, for simplification.

Figure 4.4.1a presents all polarities of 3 variables as nodes of a hypercube. We want to go through all polarities in Figure 4.4.1a. It means we go through all nodes with Hamming Distance 1 (HD1) in the hypercube graph. The black arrows show the sequence of nodes (a loop) with Hamming distance (HD) of 1 between any two neighbors. Figure 4.4.1b does the same using the KMap corresponding to this hypercube. Figure 4.4.2 illustrates that the changes are in only one bit each and the polarities are in the Gray code sequence. This is the way we exhaustively search all polarities.

As we know, there are several advantages of the exhaustive search:

a) It can be applied to find exact solutions for small functions; this is useful to create and evaluate approximate algorithms for the same task.
b) It helps to understand the structure of the problems.
c) It can be solved by classical or quantum search relatively easily.

This is why I devote so much attention to exhaustive search in my thesis.

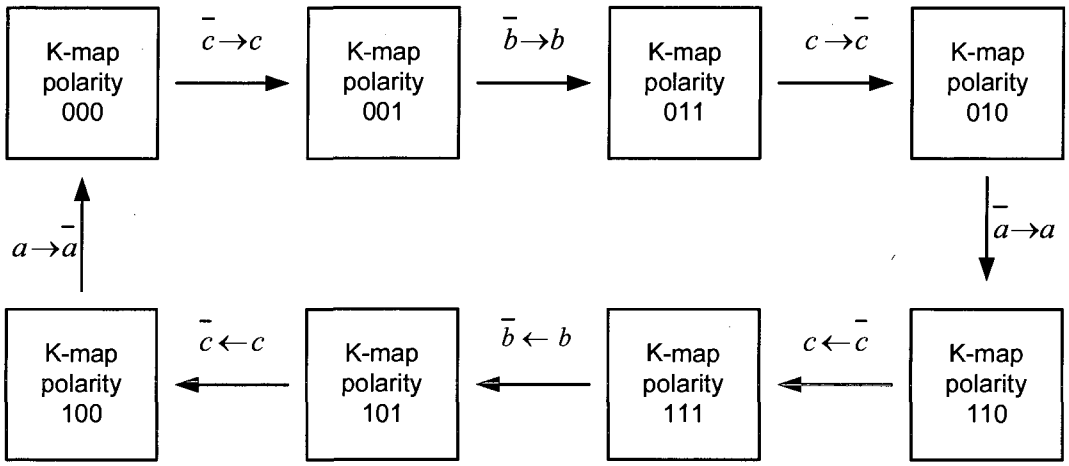


Figure 4.4.2: A sequence of HD1 polarity Exor Maps for the exact minimum FPRM generation. We start with polarity 000 and end up with polarity 100. The transformation $a \rightarrow \bar{a}$ at the right is thus not executed.

Each box in Figure 4.4.2 represents an Exor Map (or in general any functional description) in the given polarity. This Figure illustrates how the Gray code sequence from the hypercube is used to generate all polarity Exor Maps, each based on the previous Exor Map only. Another, more detailed, illustration of polarity search is given in Figure 4.4.3.

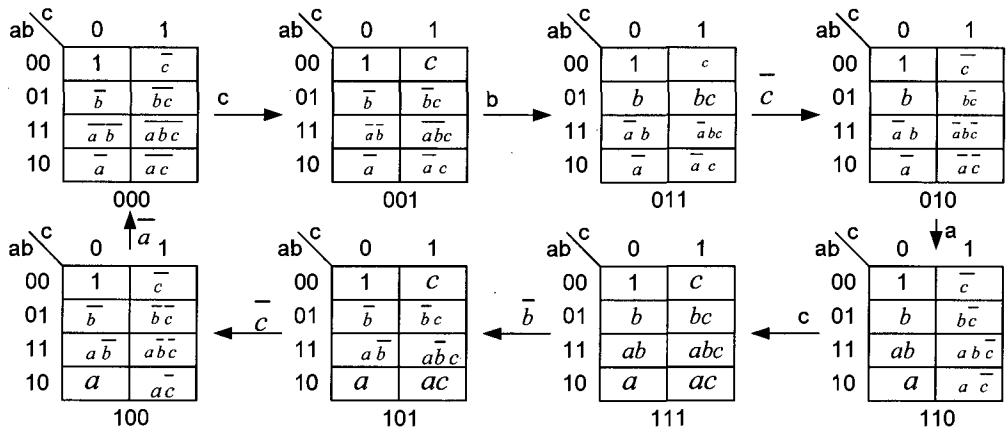


Figure 4.4.3: Graphical method to obtain function F for new polarity FPRM from the previous polarity FPRM while looping through Exor Maps of all polarities in Gray code. This Figure represents the same idea as Figure 4.4.2 and illustrates the very powerful concept of polarity search which is one of the fundaments of my thesis.

The FPRM with the smallest number of terms for forms with all these polarities is the exact minimum (term-wise) FPRM for function F.

4. 5. Tree search methods for the generation of a heuristic subset of FPRM forms.

While the method from section 4.4 generates the exact minimum FPRM, it can not be applied to large functions. Thus we explain the tree search method based on greedy (partial) search through the nodes of the hypercube of polarities.

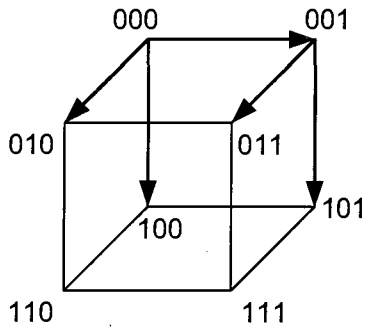


Figure 4.5.1: The (partial) tree search in the hypercube with polarities as nodes.

In section 4.4 we traversed systematically in Gray code through all polarities. But here we started from certain point 000 in Figure 4.5.1, and we find three candidate polarities (001,010 and 001) as per Gray code Hamming Distance of 1. It is the Tree search method, only for the subsets of hypercube nodes. This is the whole idea, we are not going through all polarities exhaustively as in section 4.4. If we can generate all, that method will be better, but normally using a tree search method we will generate only a subset. The search is greedy, i.e. opportunistic. Using the gradient of the cost (quality) function, we go through these sub-hypercubes where the cost function is locally minimal. Example 4.5.1 and Example 4.5.2 explain the Tree search method in details.

Example 4.5.1:

Given is function f as on top of Figure 4.5.2 in initial polarity 000. We check all HD1 polarities 001, 010 and 100. The FPRM for each of them has 8 terms. Thus we select

randomly the polarity 001. (Figure 4.6.2 shows the expansion only for this polarity) Now starting from polarity 001 (greedy search) we generate its HD1 polarities (not yet used) 011 and 101. The FPRMs in them have both costs of 4 terms. Thus any of these, say 011 is selected. The only remaining HD1 polarity of 011 is 111. The FPRM for polarity 111 is found which has 1 term.

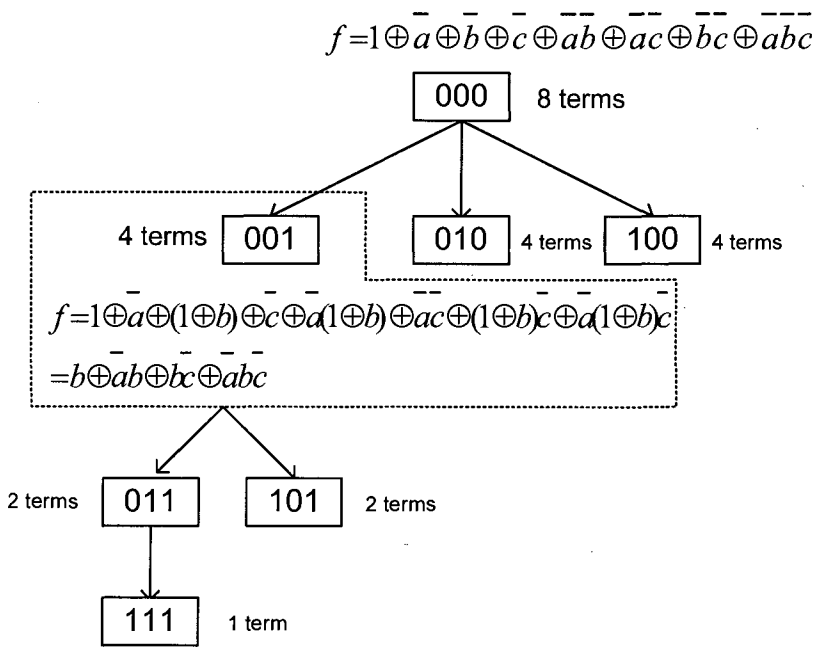


Figure 4.5.2: The Tree search corresponding to the sub tree for polarities from Figure 4.5.1. The FPRM equations for nodes 010, 100, 011, 101 and 111 are not shown to simplify the figure.

Example 4.5.2:

Given is function $f=1\oplus a\oplus b\oplus ab$.

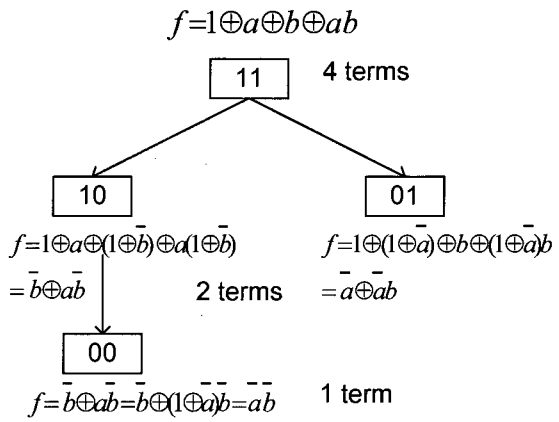


Figure 4.5.3: Tree (this time exhaustive) that visually illustrates the tree search for Example 4.5.2. The binary polarities of expansions are inside boxes. An equation is given to help the reader to analyze the tree.

The Figure 4.5.2 visually explains the Tree search method for the function $f=1\oplus\bar{a}\oplus\bar{b}\oplus\bar{c}\oplus\bar{a}\bar{b}\oplus\bar{a}\bar{c}\oplus\bar{b}\bar{c}\oplus\bar{a}\bar{b}\bar{c}$ and the Figure 4.5.3 explains visually the Tree search method for the function $f=1\oplus a\oplus b\oplus ab$. The Tree search method starts from a certain node or polarity and then systematically goes through the neighbors of that particular node (polarity). In Figure 4.5.3, we started from polarity 11 for the function $f=1\oplus a\oplus b\oplus ab$, and we are changing variable b to reach polarity node 10 and changing variable a to reach polarity node 01 as in Figure 4.5.3. The search algorithm works in one-bit changes, hence it changes one polarity to other HD1 polarity. This way our algorithm works. In Figure 4.5.3 we found that the polarity node 11 has 4 terms and for polarity node 10 and 01 the nodes have only two terms and the polarity 00 has only 1 term. So, we found the local minimum cost, the minimum number of terms in FPRM expression f = $\bar{a}\bar{b}$ realizing the function $f=1\oplus a\oplus b\oplus ab$.

Same as in Figure 4.5.2, which realizes the function $f=1\oplus\bar{a}\oplus\bar{b}\oplus\bar{c}\oplus\bar{a}\bar{b}\oplus\bar{a}\bar{c}\oplus\bar{b}\bar{c}\oplus\bar{a}\bar{b}\bar{c}$

at starting polarity node in our example is 000 with 8 terms, then it changes to polarity node 001 which have 4 terms, which means, the polarity node 001 has smaller cost. Thus we can expand other nodes as per our greedy or systematic tree search using the heuristic subset of FPRM polarities. Now we can say that if we compare the method with traversing through all FPRM forms, in the worst case, this tree search algorithm will generate all polarities (but it is not worthy). Observe that for large functions of the real life, we always want to generate the subset based on the cost function. That means, where we expect the minimum cost, but we are looking for a local minimum only.

4. 6. Evolutionary generation of FPRM forms.

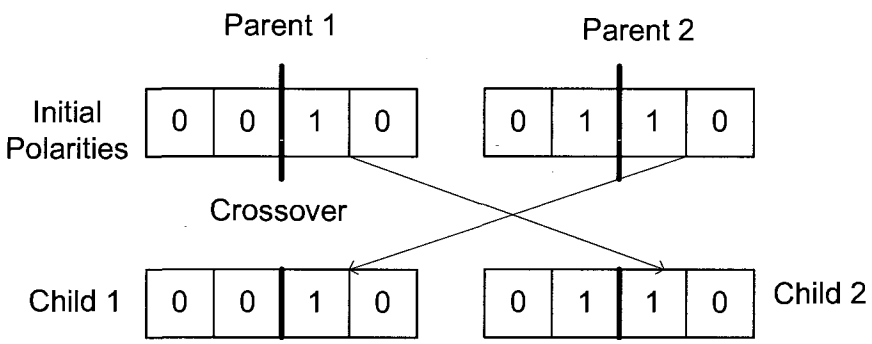


Figure 4.6.1: Evolutionary Generation of FPRM forms in various polarities. Parents and children in the genotype are polarities as binary numbers. The phenotypes are FPRM circuits corresponding to these polarities.

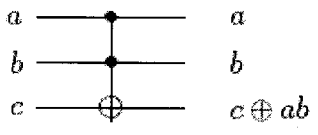
Our Evolutionary algorithm outlined in chapter 3 was randomly executing crossovers and mutations, which means, we create something, which may be a solution or not. So, our chromosomes can describe the entirely incorrect circuits. However, polarity based generation of evolutionary FPRM forms introduced here by me creates always correct solutions. Now, let's say, in Figure 4.6.1, we have the polarity 0010 of the mother chromosome and the polarity 0110 of the father chromosome. By crossover we take from mother the 00 and from father we take 10 which creates the string chromosome (0010) same as mother, nothing new. Our other child polarity (0110) is new, but it may be a polarity leading to more expensive phenotypes than the parents. When we are operating on polarities, each of the solutions is correct. Fact is that we can generate more expensive polarity. Now that the chromosomes are polarities, we do crossover and mutation same as before. But now every chromosome describes a correct circuit. Whether the cost is better or worse, we do not know. So, we still have to calculate the cost function but every solution is correct. This is the difference of our evolutionary approach with respect to the previous GA approaches. Because, if we are using Genetic algorithm to polarities as a genotype, we are always within the space of correct solutions. We are not creating nonsensical circuits. This simple idea leads in my algorithm from chapter 8 to big cost improvements for solutions.

4.7. The concept of Distance Gates.

In the new distance gate concept, we are systematically creating all possible functions in K-map for a particular Hamming distance between two minterms. It only changes the specific positions in KMap of the reversible function where we are permutating numbers, all other input vectors are the same as output vectors. This is the concept of the "distance gate". We can create pattern by changing order of variables and also by negating the inputs. Here we will explain formations of all possible distance 1 Toffoli gates. Our plan is to answer these questions:

- How many distance-1 gate exists for 3-variable functions?
- How each of them can be realized by inverting each inputs and outputs to the Toffoli gate?
- Generalize to n input functions.

Toffoli gate



ab \ c	0	1
00	000	001
01	010	011
11	111	110
10	100	101

Cube ab

Figure 4.7.1: Toffoli gate is HD1 gate permuting inside cube ab.

As seen in Figure 4.7.1, the Toffoli gate with EXOR in the bottom qubit is the distance 1 (HD1) gate for minterms 110 and 111(a cube 11X). It permutes only inside this cube keeping all other cells with no change.

Inverting input a

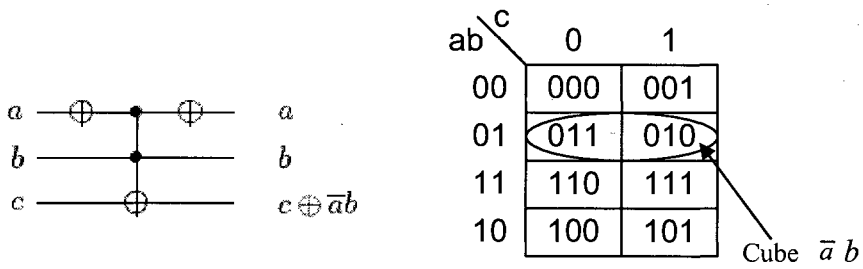


Figure 4.7.2: The Toffoli-like HD1 gate permuting inside cube $\bar{a}\,b$.

As we see in Figure 4.7.2 by inverting a single input a we still have a distance gate but the pair of affected minterms shifted to the cube 01X.

Inverting input b

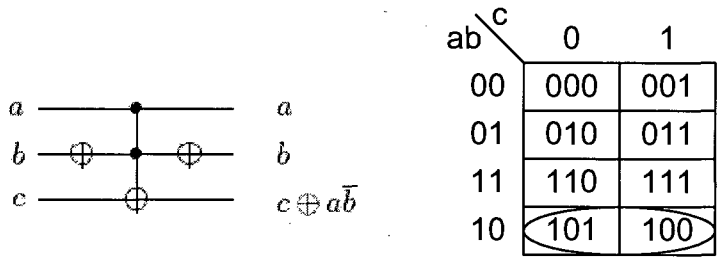
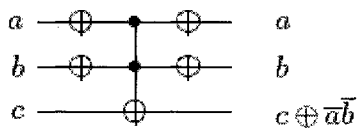


Figure 4.7.3: Toffoli-like HD1 gate permutting inside cube $a\,\bar{b}$.

Negating variable b creates the cube 10X for minterms 100 and 101 as in Figure 4.7.3.

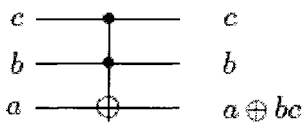
Inverting input a and b



ab \ c	0	1
00	001	000
01	010	011
11	110	111
10	100	101

Figure 4.7.4: Toffoli-like HD1 gate permutting inside cube $\bar{a}\ \bar{b}$.

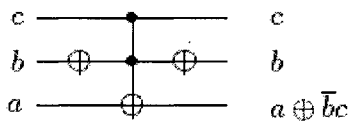
Swapping input a and c



ab \ c	0	1
00	000	001
01	010	111
11	110	011
10	100	101

Figure 4.7.5: Toffoli-like HD1 gate permutting inside cube bc.

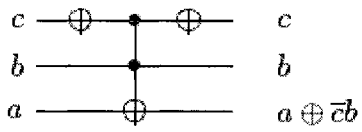
Inverting input b



ab \ c	0	1
00	000	101
01	010	011
11	110	111
10	100	001

Figure 4.7.6: Toffoli-like HD1 gate permutting inside cube $\bar{b}\ c$.

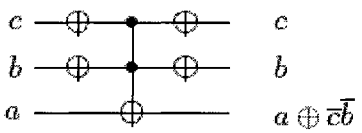
Inverting input c



ab \ c	0	1
00	000	001
01	110	011
11	010	111
10	100	101

Figure 4.7.7: Toffoli-like HD1 gate permutting inside cube $b\bar{c}$.

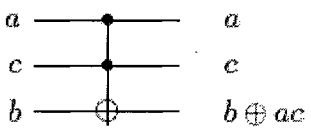
Inverting input b and c



ab \ c	0	1
00	100	001
01	010	011
11	110	111
10	000	101

Figure 4.7.8: Toffoli-like HD1 gate permutting inside cube $\bar{b}\ \bar{c}$.

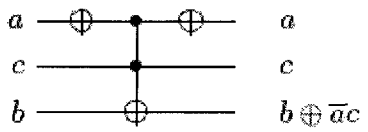
Swapping input b and c



ab \ c	0	1
00	000	001
01	010	011
11	110	101
10	100	111

Figure 4.7.9: Toffoli-like HD1 gate permutting inside cube ac.

Inverting input a



ab \ c	0	1
00	000	011
01	010	001
11	110	111
10	100	101

Figure 4.7.10: Toffoli-like HD1 gate permutting inside cube $\bar{a}\,c$.

As illustrated visually in Figures 4.7.1 - 4.7.10, HD1 gates change in specific numbers of their respective KMap only. We found the general explanation that all those circuits in Figures 4.7.1 – 4.7.10 systematically generated only by executing permutation of two cells in the KMap, all other cells remain unchanged. This means that we are always doing some local replacement of two numbers. And all other numbers remain the same. This observation is very important for our new synthesis methods (chapters 7 – 9). We call this distance one gate, Hamming distance one.

Now, the main idea is that we found HD1 the pattern in KMaps, thus we can build also a distance 2 gate for 3 variables (The HD2 gate). Which is shown below in Figure 4.7.11.

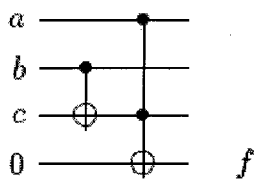


Figure 4.7.11: The gate for $f = a(b \oplus c)$ is a simple distance-2 gate (HD2 gate) with no restoration of inputs.

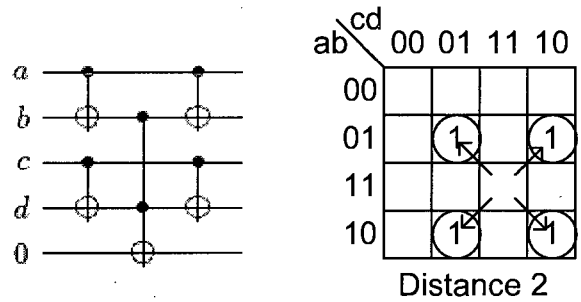


Figure 4.7.12: Toffoli gate surrounded by linear gates creates a distance 2 gate for four variables.

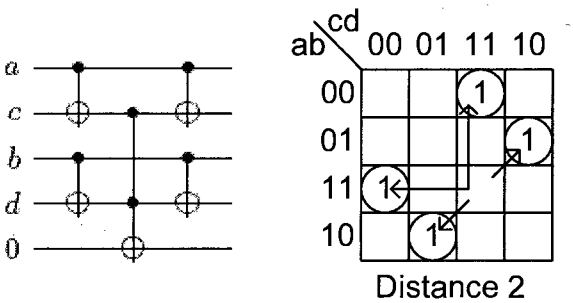


Figure 4.7.13: Changing the order of variables in Figure 4.7.12 creates another distance 2 gate as can be verified in the corresponding KMap.

This interesting distance-2 gate can be also recognized as a pattern in KMaps. Software should find these patterns in order to synthesize quantum circuits. Other, more complex example of gates with higher Hamming distances are shown in Figure

4.7.12 and Figure 4.7.13. The linear gates that surround Toffoli gate will be called pre- and post-processor and will find many applications in my thesis.

4. 8. Conclusion on generation of FPRM and similar forms.

Concluding this chapter: Based on literature, we introduced the concept of the algebraic group that is fundamental to all our synthesis methods and next we introduced the Fixed Polarity Reed Muller forms that are well known and allow to create algorithms relatively easily. These forms allow to explain the concept of polarity well, especially that this concept is not well explained in literature for higher order forms. Based on our explanation of polarity for FPRM it will be relatively easy to extend this concept for higher order AND/EXOR forms and other (Linearly Independent) forms. This will be done in chapters 7, 8 and 9. We introduced also Kronecker forms and Generalized Reed Muller forms which will be of our interest in next chapters. Finally we showed the concept of polarity Exor Maps that is a base of extending all algorithms from one polarity to all polarities. We showed that many algorithms for FPRM (and other forms) belong to three categories:

1. **Exhaustive search based on linear (ring) transversal through all polarities**. These methods are based on butterflies and polarity maps. They cannot be applied to larger circuits, but the concepts of these algorithms can be applied to create both sequential (standard, classical), parallel and quantum architectures.

They explain also the structure of such families.

2. **Tree-Search methods that use a tree to generate all or some solutions.** These methods are linked to the main methods of this thesis and allow to create both exact and more importantly efficient heuristic (approximate) search algorithms.

3. **Evolutionary algorithms.** While a ring or a tree are basis "generation structures" in the above categories of algorithms, the evolutionary algorithms are chaotic and they have no structure. Instead of going through the solution space systematically in linear or tree-like fashion, they jump from point to point (polarity to polarity) in a pretty random way. However, they are supposed to find solutions based on fitness function. So far, the reality did not confirm superiority of these algorithms in any area related to quantum permutative circuits, but they are still one of the best for the non-permutative quantum circuits specified by arbitrary matrices [Lukac04].

The comparison and combination of three families of search algorithms; linear, tree-like and evolutionary is the fundament of universal hybrid search methods developed in this dissertation.

Finally we introduced the new concept of Distance gates for various Hamming

Distances of permuted cells. Such gates will be used in algorithms and further generalized. Now we are thus ready to learn about more advanced families of forms and try to apply these three types of search algorithms to them. This will start in chapter 7 where iterative deepening search is discussed, chapter 8 where the generated search is applied to GRM and other forms and chapter 9 where these methods are extended to Linearly Independent families. Chapters 5 and 6 are devoted to universal search methods, both classical and quantum, sequential and parallel, developed for the class of combinatorial CAD problems that are of interest to us.

CHAPTER 5

Quantum Algorithms

5.1. Introduction. Classes of Oracles for Grover Algorithm

It is well-known that a quantum algorithm can be orders of magnitude more efficient than a standard algorithm. Unfortunately, there are very few algorithms in addition to the famous Shor [Shor94] and Grover [Grover96] algorithms that are known to be of any practical use outside pure mathematics. One has thus to consider how the known quantum algorithms can be used to solve practical problems. There is a large class of highly complex CAD problems that cannot be exactly solved on standard computers. But they could be solved if a classical Satisfiability or one of other similar Decision Functions were solvable. This can be done using Grover Algorithm.

This chapter is based on literature with the exception of the graphical analysis method and simulation example. This chapter will present quantum algorithms: Deutsch, Deutsch-Jozsa, Bernstein-Vazirani and their modifications and next the Grover algorithm. Next, in chapters 6, 7 these ideas will be related to hierarchical parallel search system and in chapter 12 a discussion of satisfiability oracles will follow and several such oracles will be constructed, including POS satisfiability, covering problems and ESOP minimization. Chapter 13 discusses oracles for graph coloring and chapter 14 the oracles for other constraint satisfaction problems. Path problems and oracles will be also discussed in the final version of the thesis. Finally, chapter 15

presents problems and oracles for spectral methods and machine learning and an overview of constraint satisfaction problems in robotics.

5.2. Quantum Algorithms

5.2.1. Introduction to Quantum Algorithms.

When we look into the laws of Nature, we can say that managing information and logic, hence computing can not exist in a detachment. That means information must be recorded/written on some physical substance such as paper, or magnetic media or neural connections in our brain, or a beam photons or electrons trapped in quantum dots. Here the physical law of these media determines the logic to store information, and to compute that information. Quantum mechanics governs the behavior and the properties of those media in a fundamental way on the microscopic scale of atoms and molecules. So, we can say that classical computers are following the rules of quantum mechanics. But the architectures of classical computers are not based on quantum mechanics. The information in computing, how the zero and the one bits evolve inside those machines can be explained by classical logic and information theory. However, quantum computers exploit the phenomena of superposition and entanglement which are fundamental issues in quantum mechanics [Nielsen00]. Thus quantum computers have additional features than their counterpart classical computers lack. Hence quantum computer are more powerful than the classical computers or in some

problems at least similar. Observe that Quantum and Classical computer both in principle can emulate each other, but with very different efficiencies.

Now, we can ask ourselves why we need Quantum Computing? Is it for the sake of flourishing and very attractive research area [Hirvensalo01] or "mathematical wishful thinking" [Manin99]? "Quantum Computing" comprises theories, algorithms and techniques for exploiting the unique nature of quantum events to obtain computational advantages. Actually that is not the reason, the fact is that the quantum computer promises great future for computing: it significantly reduces the times of solving many computational tasks. It was shown by Peter Shor in 1994 that the problem of factoring a number into prime numbers could efficiently be solved on a quantum computer in polynomial time [Shor94], hence showing the advantage of Quantum computers in solving efficiently problems that are hard for classical computers. (Hard means that the time for solving the problem grows at least exponentially with the length of input data). Again, a classical simulation of quantum mechanic problems typically suffers from exponential slowdown, whereas quantum system could in principle execute the simulation of any other quantum system efficiently [Feynman96]. Apart from the computational power, Moore's law has physical limits [Moore65]; then in year 2020 the components of computers will be on atomic scale where quantum effects are dominant. So, quantum computers will be one of the natural solutions for future computing. Moreover, we know from Landauer [Landauer61] that binary logic circuits built using irreversible gates lead inevitably to energy dissipation, regardless of the technology used to realize the gates. Zhirnov *et al.* [Zhirnov03] showed that power

dissipation in any future CMOS will lead to impossible heat removal and thus the speeding-up of CMOS devices will be impossible at some point which will be reached soon. Bennett [Bennett73] proved that for power not to be dissipated in a binary logic circuit, it is necessary that the circuit be built from reversible gates. Thus all output patterns are just permutations of input patterns. Such circuit can be described by a permutation matrix. Bennett's theorem suggests that every future binary technology will have to use some kind of reversible gates in order to reduce power dissipation. Quantum technology is inherently reversible and is now the most promising technology for future computing systems [Nielsen00]. Even small building blocks of a quantum computer or small quantum circuits may be useful like in quantum cryptography [Gisin02], in atomic clock [Wineland94, Huelga97], in entanglement distillation [Bennett96, Horodecki01]. (In the atomic clock, a quantum circuit could in principal reduce the uncertainty of the clock by a factor $\sqrt{N}$ by generating quantum correlations between N relevant atoms, which is very important in global positioning system and in synchronizing networks and distant telescope. Besides, application of small quantum circuits in many cases distills a few highly entangled states out of many weakly entangled ones in order to distribute entangled states over large distances. We have to send them through inevitably noisy channels, thereby loosing some of the entanglement.)

Moreover, Quantum Circuits (QC) have an advantage of being able to perform massively parallel computations in one time step [Hirvensalo01] which causes interest in them to perform in future massively parallel computations.

5.2.2. Background

This section contains preliminary background discussion of basic topics of binary quantum algorithms and Grover's Algorithm.

As we have shown in chapters 2 – 4 the circuit model for the quantum computer [Nielsen00] is in principle very similar to the traditional circuit model. We know that a classical computer operates on a vector of input bits and returns a vector of output bits, hence the functions can be described as logical circuits built out of many elementary logic operations. Thereby, in quantum computers we have to replace the input-output function by a quantum operation mapping. In quantum computing bits are replaced by qubits in a complex multi-dimensional Hilbert space; these spaces and their tensor products constitute the objects of quantum algebra as well.

5.2.3. Quantum Oracle.

An *oracle* is a logic circuit that answers "yes/no" to a question asked to it, for instance – "is this mapping of nodes to colors a correct graph coloring?" Quantum oracles are built from truly quantum gates and many oracles include arithmetic, logic, and mixed blocks. The oracle architecture is very suitable for quantum computers and basically, it

is one of very few architectures investigated in this field. We know the probabilistic read-out nature of a quantum system, thus if one only knows how to build a respective oracle, Grover algorithm and its modifications would be immediately useful to solve many problems when the physical quantum computers will become available. These algorithms are either deterministic or probabilistic in nature. It is therefore important to study methods and algorithms to build various types of oracles. The problem of building various classes of oracles or their blocks (components) is well known in case of binary quantum circuits [Nielsen00, Perkowski04]. The questions asked the oracles may be as elaborate as you can make them, the procedure that answers the questions may be lengthy and a lot of auxiliary data may get generated while the question is being answered. Yet all that comes out is just *yes* or *no*. However, an oracle is the portion of an algorithm which can be regarded as a "black box" whose behavior can be relied upon. Figure 5.2.3.1 illustrates the nature of quantum oracle, input to the synthesis is a black box for a function *f(x)* that verifies some property.

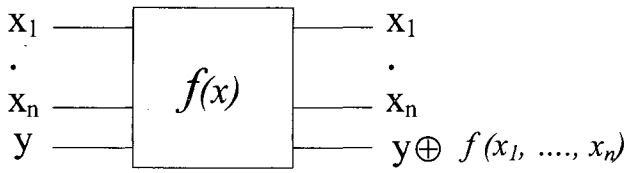


Figure 5.2.3.1: Oracle for quantum algorithms.

This oracle architecture is very suitable for quantum computers, for simplification, even a single quantum gate can be treated in some problems as a black box, i.e. Quantum Oracle. We remember probabilistic nature of Quantum system measurement.

This is an important research and experimentation problem. However, in this thesis the challenges are to implement practical quantum computers and to design quantum circuits made of available quantum gates [Deutsch89].

The quantum algorithms that will be explained as an introduction to Grover algorithm are the following:

The Deutsch Algorithm

This algorithm answers the following question. Suppose we have a function $f:\{0,1\}\rightarrow\{0,1\}$, which can be either *constant* or *balanced.* In this case the function is constant if $f(0) = f(1)$ and it is balanced if $f(0)\neq f(1)$. Classically it would take two evaluations of the function to tell whether it is one or the other. Quantumly, we can answer this question in a single evaluation only. The reason for this is that quantumly we can pack 0 and 1 into x at the same time, of course. This fact was the first breakthrough in quantum computing thanks to David Deutsch.

The Deutsch-Jozsa Algorithm

This algorithm generalizes the Deutsch algorithm to a function $f:\{0,1\}^n\rightarrow\{0,1\}$. We ask the same question: is the function is constant or balanced. Here *balanced* means that the function is 0 on half of its arguments and 1 on the other half. Of course in this case the function may be *neither* constant nor balanced. In this case the oracle doesn't work: it may say *yes* or *no* and the answer will be meaningless. Although deeper than Deutsch algorithm, this extension by Jozsa was still limited to particular uses.

The Bernstein-Vazirani Algorithm

Suppose there is a function $f:\{0,1\}^n \to \{0,1\}$ of the form $f(x) = a \bullet x$, where **a** is a constant vector of 0s and 1s and - is a scalar product, x is vector of input variables, thus f(x) is a linear Boolean function always, for instance $x_1 \oplus x_2 \oplus x_3$. How many measurements are required to find the value of **a**? Classically one has to perform measurements for all possible arguments and then solve a system of linear equations for **a**. Quantumly **a** is delivered in one computational step on output lines of the oracle. This problem has limited but practical applications in logic synthesis and image processing. It is related to Walsh-Hadamard spectral transforms which find many applications.

The Simon Algorithm

Suppose there is a function $f:\{0,1\}^n \to \{0,1\}^n$. The function is supposed to be 2-to-1, i.e., for every value of f there are always two n-arguments such $\mathbf{x}_1$ and $\mathbf{x}_2$ that $f(\mathbf{x}_1) = f(\mathbf{x}_2)$. The function is also supposed to be periodic, meaning that there is such a binary vector **a** that $f(\mathbf{x} \oplus \mathbf{a}) = f(\mathbf{x})$, where $\oplus$ is a bitwise EXOR on words $\mathbf{x}_1$ and $\mathbf{x}_2$. The algorithm returns the period **a** in *O(n)* measurements. Of course, if one disposes a sufficiently large ensemble of quantum computers then a single computation will return the answer in the density operator, but we are not discussing ensemble computers much in this thesis. This algorithm is historically very important as it started Shor to think about using periodicity which led ultimately to the discovery of the Shor algorithm.

5.2.3.1. The Deutsch Algorithm

The circuit that implements the Deutsch Oracle is shown in Figure 5.2.3.1.1:

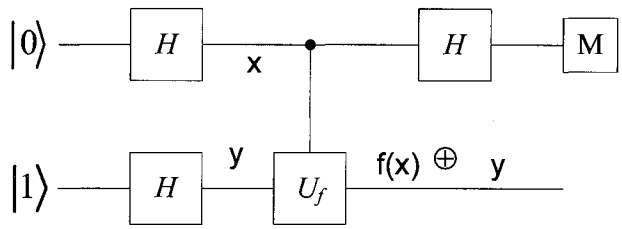


Figure 5.2.3.1.1: Deutsch Quantum Algorithm. M is the single-qubit measurement operator. f is a one-argument Boolean function.

Here **H** is the Hadamard gate, which we have already encountered in Chapter 2:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

or in matrix notation:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

In Figure 5.2.3.1.1 the block U_f denotes a controlled gate defined as follows, using Dirac notation:

$$U_f|xy\rangle = |x\rangle \otimes |y \oplus f(x)\rangle$$

Function f maps {0, 1} to {0, 1}. Symbol $\otimes$ is the tensor product. Function of f can be either constant or balanced. As discussed at the beginning of this section, on a

classical computer two measurements are required to figure out one if function f is balanced or constant.

The Hadamard gate in the upper qubit converts $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$. Thus both $|0\rangle$ and $|1\rangle$ are simultaneously given to x.

The detailed analysis of this circuit will follow:

1. The first pair of Hadamard gates performs the following transformation:

$$|01\rangle \rightarrow \frac{1}{2}(|0\rangle+|1\rangle)\otimes(|0\rangle-|1\rangle)$$

2. Now the controlled-U_f gate is applied to this quantum state. The Equation 5.2.3.1.1 is now derived:

$$U_f|x\rangle\otimes(|0\rangle-|1\rangle)=|x\rangle\otimes((|0\rangle-|1\rangle)\oplus f(x)) \qquad \textit{Equation 5.2.3.1.1}$$

Observe that when $f(x) = 0$ then the Equation 5.2.3.1.2 holds.

$$(|0\rangle-|1\rangle)\otimes f(x)=|0\rangle-|1\rangle=(-1)^0(|0\rangle-|1\rangle)=(-1)^{f(x)}(|0\rangle-|1\rangle) \qquad \textit{Equation 5.2.3.1.2}$$

Similarly for $f(x) = 1$ we have Equation 5.2.3.1.3.

$$(|0\rangle-|1\rangle)\otimes f(x)=|1\rangle-|0\rangle=(-1)^1(|0\rangle-|1\rangle)=(-1)^{f(x)}(|0\rangle-|1\rangle) \qquad \textit{Equation 5.2.3.1.3}$$

It results from Equation 5.2.3.1.2 and Equation 5.2.3.1.3 that the same formula holds

for all values of $f(x)$. This leads to Equation 5.2.3.1.4.

$$U_f|x\rangle\otimes(|0\rangle-|1\rangle)=(-1)^{f(x)}|x\rangle\otimes(|0\rangle-|1\rangle) \qquad \textit{Equation 5.2.3.1.4}$$

Applying equation to our state from step1 leads to Equation 5.2.3.1.5.

$$\begin{aligned} &U_f\frac{1}{2}(|0\rangle+|1\rangle)\otimes(|0\rangle-|1\rangle) \\ &=\frac{1}{2}\left((-1)^{f(0)}|0\rangle+(-1)^{f(1)}|1\rangle\right)\otimes(|0\rangle-|1\rangle) \end{aligned} \qquad \textit{Equation 5.2.3.1.5}$$

3. Finally the Hadamard gate is applied to the first vector as in Equation 5.2.3.1.6.

$$(-1)^{f(0)}|0\rangle+(-1)^{f(1)}|1\rangle \qquad \textit{Equation 5.2.3.1.6}$$

Which leads to Equation 4.2.3.1.7.

$$\begin{aligned} &\frac{1}{2}\left((-1)^{f(0)}H|0\rangle+(-1)^{f(1)}|1\rangle\right)\otimes(|0\rangle-|1\rangle) \\ &=\frac{1}{2}\left((-1)^{f(0)}\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)+(-1)^{f(1)}\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right)\otimes(|0\rangle-|1\rangle) \\ &=\frac{1}{2}\left(|0\rangle\left((-1)^{f(0)}+(-1)^{f(1)}\right)+|1\rangle\left((-1)^{f(0)}-(-1)^{f(1)}\right)\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \end{aligned}$$

Equation 5.2.3.1.7

When $f(x)$ is a constant function then $(-1)^{f(0)} - (-1)^{f(1)} = 0$. This means that Equation 5.2.3.1.7 reduces to Equation 5.2.3.1.8 for the upper qubit.

$$\frac{1}{2}\left(|0\rangle\left((-1)^{f(0)}+(-1)^{f(1)}\right)\right)=\pm|0\rangle \qquad \textit{Equation 5.2.3.1.8}$$

When $f(x)$ is balanced then $(-1)^{f(0)} + (-1)^{f(1)} = 0$ which reduces the upper qubit to Equation 5.2.3.1.9.

$$\frac{1}{2}\left(|1\rangle \left((-1)^{f(0)} - (-1)^{f(1)}\right)\right) = \pm|1\rangle \qquad \textit{Equation 5.2.3.1.9}$$

Thus to find if function $f(x)$ is constant or balanced we have to measure the upper qubit vector. If it is $|0\rangle$ then $f(x)$ is a constant, if it is $|1\rangle$ then $f(x)$ is a balanced function.

One may wonder what happens to the bottom qubit and why the values such as $(-1)^{f(x)}$ mysteriously shifted to the upper qubit and have not stayed with the bottom qubit. The answer is that the bottom qubit is allowed to decohere (as a garbage qubit). As it does so, it collapses onto $|0\rangle$ or $|1\rangle$, thus forcing the parameters that describe the bipartite state onto the upper qubit. The Deutsch oracle is a very nice and simple demonstration of the essentials of quantum computing: first it shows the power of quantum parallelism, then it shows the importance of entanglement and non-locality in quantum computing. Every quantum computer demonstrates that the quantum states perceived as nonsensical by the Einstein-Podolsky-Rosen paradox truly exist, thus proving Einstein to be wrong [Einstein35, Bennett93] and demonstrating that true science has no authorities, only facts.

Now I will explain the Deutsch Algorithm one more time using a simple transformation based method. The oracles for cases f (x) = 0, f (x) = 1, f (x) = x and f (x) = $\overline{x}$ are shown in Figure 5.2.3.1.2.

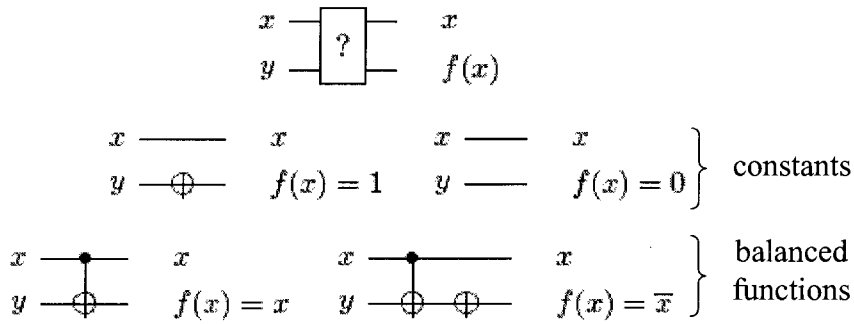


Figure 5.2.3.1.2: Four cases of the Deutsch oracle, function f(x)=0 and f(x)=1 are constants, function f(x)=x and f(x)= $\bar{x}$ are balanced.

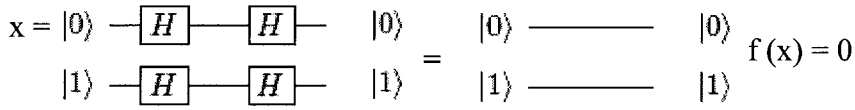


Figure 5.2.3.1.3: Oracle with input and output Hadamards for the case f(x) = 0.

Using quantum equivalence transformations from chapter 2 the left part of Figure 5.2.3.1.3 is transformed to the right part. As expected, the upper qubit is |0⟩ before measurement so it will be 0 after the measurement. Thus for f(x)=0 a constant, the measured qubit is 0. Similarly, analyzing case of f(x) = 1, Figure 5.2.3.1.4, the quantum equivalence transformation produces the circuit from the right which gives 0 on the first qubit after measurement.

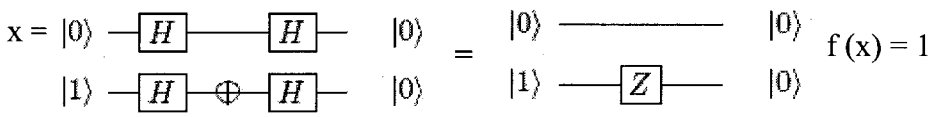


Figure 5.2.3.1.4: Oracle with input and output Hadamards for the case f(x) = 1.

Using the quantum equivalence transformation the circuit (algorithm) for case f(x) = x, (Figure 5.2.3.1.5 left) is converted to the circuit from the right of Figure 5.2.3.1.5 thus giving value 1 in the measurement of the upper bit for balanced function f(x) = x.

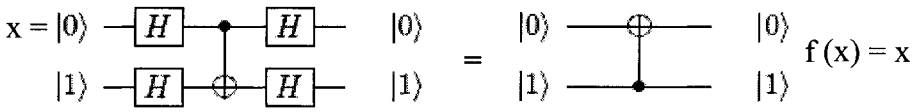
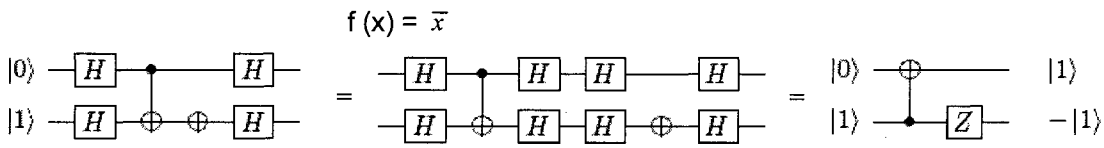
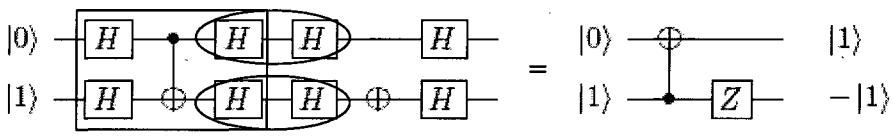


Figure 5.2.3.1.5: Oracle with input and output Hadamards for the case f(x) = x.



(a)



(b)

Figure 5.2.3.1.6: Oracle with input and output Hadamards for the case f(x) = $\bar{x}$. (a) stages of transformation, (b) marked sub circuits subject to transformations in Figure 5.2.3.1.6a.

Transformation of the last case of the algorithm, when f(x) = $\bar{x}$ is more tricky. The original circuit is given at the left in Figure 5.2.3.1.6. We can not apply any meaningful transform to simplify this circuit directly. But we can see that the upper and lower wires directly to the right of the Feynman gate can be replaced as a serial composition of the Hadamards each. This allows to apply the transform from Figure

5.2.3.1.5 to the left part of the circuit (in a rectangle). At the same time the transformation from Figure 5.2.3.1.4 is applied to the right part of the left circuit from Figure 5.2.3.1.6b thus leading to the right circuit from Figure 5.2.3.1.6b. Clearly, when we measure the upper qubit it will be a "1". Thus, the measurement of the upper qubit is always 0 for constant functions and always 1 for balanced functions. The same result as derived analytically by Deutsch. The graphical method here shows also that the choice of a particular transformation rule, is non-trivial, as we have to make first a more complex circuit by adding two pair of Hadamards, in order to be able to simplify it later on by applying quantum equivalence transformations. I hope that a general intelligent smart software based on such transformations will prove some interesting facts.

5.2.3.2. The Deutsch-Jozsa Algorithm

Jozsa extended Deutsch ideas so his oracle is similar to the Deutsch oracle, but it has more bits:

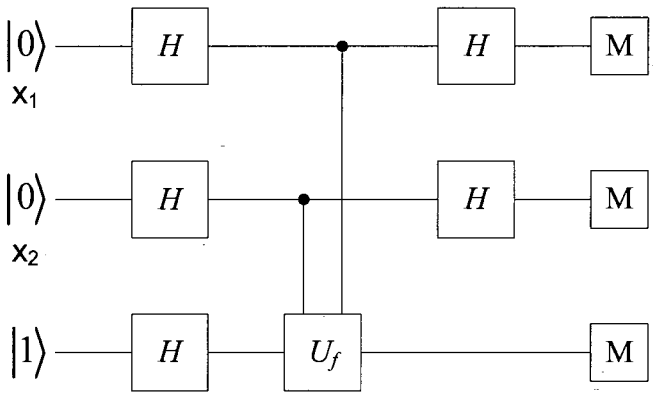


Figure 5.2.3.2.1: Deutsch-Jozsa Quantum Algorithm with two inputs x_1 and x_2 as measurement of f .

The role of the Hadamard gates is the same as in Deutsch algorithm. However, the U_f gate is now controlled by n qubits (by two qubits x_1 *and* x_2 *in Figure 5.2.3.2.1).* Function f maps from $\{0, 1\}^n$ to $\{0, 1\}$. As in Deutsch algorithm function f can be either *constant* or *balanced.* The task of the quantum circuit is to check whether U_f is a constant or a balanced function by performing just one measurement. A classical oracle would require 2^n measurements, one for each value of the argument, to ascertain that f is constant. By knowing that the choice is only between constant and balanced functions the classical oracle would still need $2^{n-1} + 1$ measurements (because by taking randomly 2^{n-1} measurements and having a 0 always we still would not know if the $2^{n-1} + 1$ measurement will give a 0 or a 1. If it will give a zero, the function is a constant, otherwise the function is balanced).

The detailed analysis of the Deutsch-Jozsa algorithm will follow.

1. As before, H gates are first applied, as in Equation 5.2.3.2.1 to n states $|0\rangle$.

$$
\begin{aligned}
& H|0\rangle H|0\rangle \cdots H|0\rangle \\
&= \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\cdots\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle) \\
&= \frac{1}{2^{n/2}}(|00\cdots0\rangle+|00\cdots1\rangle+\cdots+|11\cdots1\rangle) \\
&= \frac{1}{2^{n/2}}(|0\rangle+|1\rangle+|2\rangle+\cdots+|2^n-1\rangle) \\
&= \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle
\end{aligned}
$$

Equation 5.2.3.2.1

Applying H to the bottom qubit gives $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. From which the state of the entire computer is described as in Equation 5.2.3.2.2:

$$\frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}|x\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \qquad \textit{Equation 5.2.3.2.2}$$

2. Now the *n*-qubit controlled U_f operator (gate) is applied. By extending the approach from the previous section, we obtain Equation 5.2.3.2.3.

$$\frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \qquad \textit{Equation 5.2.3.2.3}$$

3. Finally the Hadamard transform is applied to the top *n* qubits again. But the top qubits are no longer just $|0\rangle$. Observe that the basic definition of Hadamard transform can be represented in Equation 5.2.3.2.4 and Equation 5.2.3.2.5.

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle) = \frac{1}{\sqrt{2}}((-1)^{0\bullet 0}|0\rangle+(-1)^{0\bullet 1}|1\rangle) \qquad \textit{Equation 5.2.3.2.4}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) = \frac{1}{\sqrt{2}}((-1)^{1\bullet 0}|0\rangle+(-1)^{1\bullet 1}|1\rangle) \qquad \textit{Equation 5.2.3.2.5}$$

Combining together Equation 5.2.3.2.4 and Equation 5.2.3.2.5 we obtain Equation 5.2.3.2.6.

$$H|x\rangle = \frac{1}{\sqrt{2}}\left(\sum_{y=0}^{1}(-1)^{x\bullet y}|y\rangle\right)$$ *Equation 5.2.3.2.6*

The Hadamard transform from Equation 5.2.3.2.6 is now applied to the tensor product of *n* qubits, leading to Equation 5.2.3.2.7.

$$\begin{aligned}
&H|x_1\rangle \oplus H|x_2\rangle \otimes \cdots \otimes H|x_n\rangle \\
&= \left(\frac{1}{\sqrt{2}}\sum_{y_1=0}^{1}(-1)^{x_1\bullet y_1}|y_1\rangle\right)\otimes\left(\frac{1}{\sqrt{2}}\sum_{y_2=0}^{1}(-1)^{x_2\bullet y_2}|y_2\rangle\right)\otimes\cdots \\
&\otimes\left(\frac{1}{\sqrt{2}}\sum_{y_n=0}^{1}(-1)^{x_n\bullet y_n}|y_n\rangle\right) \\
&= \frac{1}{2^{n/2}}\sum_{y_1 y_2\cdots y_n}(-1)^{x_1\cdot y_1}(-1)^{x_2\bullet y_2}\cdots(-1)^{x_n\bullet y_n}|y_1\, y_2\cdots y_n\rangle \\
&= \frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{x\bullet y}|y\rangle
\end{aligned}$$ *Equation 5.2.3.2.7*

where $x.y = x_1.y_1 + x_2.y_2 + \cdots + x_n.y_n$. Observe that the addition is here arithmetic, not Boolean or modulo. The final expression from Equation 5.2.3.2.7 is now inserted into our formula. This leads to Equation 5.2.3.2.8.

$$\frac{1}{2^{n/2}}\left(\frac{1}{\sqrt{2}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\overset{n}{\otimes}H|x\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$$
$$=\frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}\frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{x\bullet y}|y\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \qquad \textit{Equation 5.2.3.2.8}$$
$$=\frac{1}{2^n}\left(\sum_{x=0}^{2^n-1}\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\bullet y}|y\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$$

Several interesting properties can be now derived from the final formula. For instance, if *f(x)* is constant, it can be taken before the sum symbol. Therefore the sum becomes as in Equation 5.2.3.2.9.

$$\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}(-1)^{x\bullet y}|y\rangle \qquad \textit{Equation 5.2.3.2.9}$$

Now, the value of $|y\rangle$ is fixed to analyze Equation 5.2.3.2.10.

$$\sum_{x=0}^{2^n-1}(-1)^{x\bullet y}|y\rangle \qquad \textit{Equation 5.2.3.2.10}$$

In case of $y \neq 0$ we obtain Equation 5.2.3.2.11.

$$\sum_{x=0}^{2^n-1}(-1)^{x\bullet y} = 0 \qquad \textit{Equation 5.2.3.2.11}$$

This is beacuse $x \bullet y$ will ``push as often to the right as to the left". So the only term that is going to survive in this case is for $\mathbf{y} = |y\rangle$. Therefore the final state of the oracle is in this case as in Equation 5.2.3.2.12.

$$\frac{1}{2^n}(-1)^{f(x)}\left(\sum_{x=0}^{2^n-1}(-1)^{x\bullet 0}|0\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) = (-1)^{f(x)}|0\rangle\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \quad \textit{Equation 5.2.3.2.12}$$

In the second case, when $f(x)$ is balanced then $|y\rangle = |0\rangle$ leads to Equation 5.2.3.2.13.

$$\sum_{x=0}^{2^n-1}(-1)^{f(x)}\,(-1)^{x.0} = \sum_{x=0}^{2^n-1}(-1)^{f(x)}|0\rangle = 0 \qquad \textit{Equation 5.2.3.2.13}$$

Observe that $f(x)$ pushes as often to the right as it pushes to the left, because f is balanced. Therefore the probability amplitude of finding $|y\rangle$ in $|0\rangle$ is zero.

Concluding these cases if function $f(x)$ is constant then measuring the output control qubits *must* return $|0\rangle$ on every qubit $x_1, \ldots, x_n$. If this is not the case then function $f(x)$ must be balanced.

Now I will use the graphical method introduced already at the end of the previous section to explain the Deusch-Jozsa algorithm for U_f controlled by 2 qubits (Figure

5.2.3.2.2). This figure shows only one case, other can be done similarly. In this case outputs in x_1, x_2 are not 0, so function is balanced.

Observe that with the graphical method I can only check quickly many cases of balanced functions that I can not make a general proof. My method still remains useful as it allows for fast testing if some property of a quantum algorithm is true.

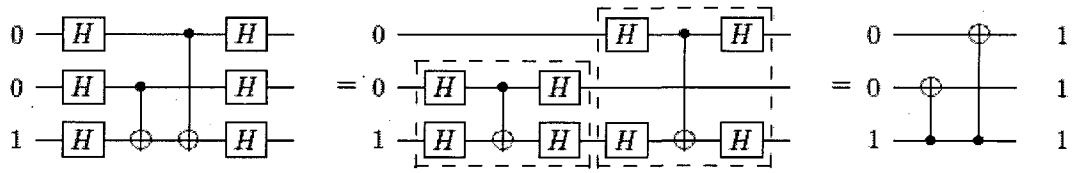


Figure 5.2.3.2.2: Graphical method applied to an instance of Deutsch-Jozsa algorithm (a) Original circuit with balance function $f(x_1,x_2)=x_1 \oplus x_2$, (b) adding two Hadamards in series to allow double applications of transformation from Figure 5.2.3.1.5, (c) The final circuit.

Although the algebraic proof is more general, the graphical method developed by me is more intuitive, especially for the digital design engineers, who are familiar with graphical transformations of logic schemata.

5.2.3.3. The Bernstein-Vazirani Algorithm

The Bernstein Vazirani Oracle is the Deutsch Jozsa oracle with $f(x) = a \bullet x$, where the multiplication is arithmetical.The final state of the oracle is described by Equation 5.2.3.3.1.

$$\frac{1}{2^n}\left(\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}(-1)^{a\bullet x}(-1)^{x\bullet y}|y\rangle\right)\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$$ *Equation 5.2.3.3.1*

Using the same method as in previous sections we write the formula for the sum over x, as given in Equation 5.2.3.3.2.

$$\sum_{x=0}^{2^n-1}(-1)^{a\bullet x}\ (-1)^{x\bullet y}|y\rangle$$ *Equation 5.2.3.3.2*

In case when $a \neq y$ this leads to value 0, because the components of the sum will push as much to the right as they will push to the left. In case when **a** = **y** the Equation 5.2.3.3.3 is obtained.

$$(-1)^{a\bullet x}\ (-1)^{a\bullet x}|y\rangle = 1$$ *Equation 5.2.3.3.3*

From Equation 5.2.3.3.3 we derive the formula for the final quantum state, as given in Equation 5.2.3.3.4.

$$\left(\sum_{y}\delta_{a,y}|y\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)=|a\rangle\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$$ *Equation 5.2.3.3.4*

Therefore the value of a is returned by measuring the control qubits.

5.2.3.4. The Simon Algorithm

An example of a Simon algorithm for a function $f:\{0,1\}^3 \rightarrow \{0,1\}^3$ is shown Figure 5.2.3.4.1.

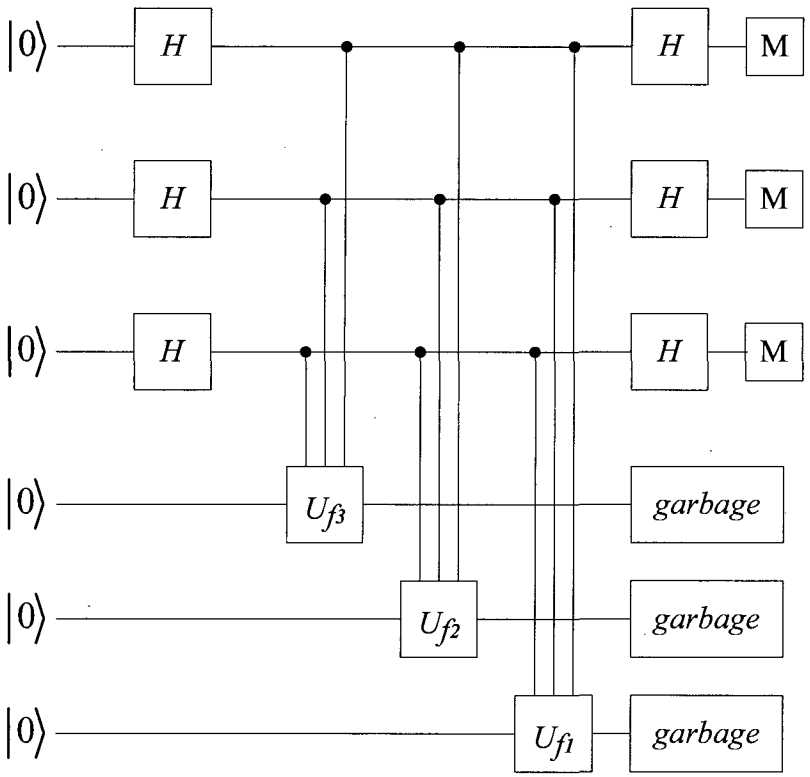


Figure 5.2.3.4.1: The Simon Algorithm.

In general the oracle comprises n qubits at the top, which look the same as the top qubits in oracles from previous sections of this chapter. Then we have n qubits at the bottom. Each of these n qubits corresponds to a sub-function $f_i:\{0,1\}^n \rightarrow \{0,1\}$, n of which make up function f.

The boxes labelled U_{fk} are the controlled-NOT gates, where the control is provided by $f_k(x)$.

In summary, the Simon Algorithm tests a function $f:\{0,1\}^n \to \{0,1\}^n$ which in Figure 5.2.3.4.1 was decomposed into n scalar-valued functions $f_k:\{0,1\}^n \to \{0,1\}$.

The oracle function in Simon Algorithm must satisfy the following conditions:

1. f is 2-to-1, i.e., for every value of f there are always two different vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ such that $f(x_1) = f(x_2)$

2. f is periodic, i.e., there exists such vector **a** that $f(x \oplus a) = f(x)$

Of course, the following question may be asked: if f is periodic then it should be more than just 2-to-1, because

$f(x \oplus a \oplus a) = f(x \oplus 0) = f(x)$

But remember that here we work within binary arithmetic and $\oplus$ is a modulo-2 addition (or EXOR), hence for every vector a we have $a \oplus a = 0$ and therefore $x \oplus a \oplus a$ takes us back to x.

Assuming that function $f(\mathbf{x})$ satisfies these conditions, the oracle presents its period a in $O(n)$ measurements.

This is a considerable improvement on a classical system designed to do the same. To find the value of **a** the classical oracle would have to be queried an exponential number of times in n.

The detailed analysis of the Simon Algorithm follows:

Step 1. Applying the Hadamard transform to the top n qubits works the same as in the Deutsch-Jozsa algorithm. We thus reuse the result obtained in step 1 of the analysis of the Deutsch-Jozsa circuit. Thus we have Equation 5.2.3.4.1.

$$\frac{1}{2^{n-1}}\left(\sum_{x=0}^{2^n-1}|x\rangle\right)\otimes|0\rangle|0\rangle\cdots|0\rangle = \frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}|x\rangle\right)\otimes|0\rangle \qquad \textit{Equation 5.2.3.4.1}$$

Step 2. The application of the U_{fk} gates at this stage converts the n bottom qubits that carry $|0\rangle$ into $|f_k(x)\rangle$. Observe in the circuit that for every individual $|0\rangle$ qubit, if its corresponding $f_k(x)$ evaluates to 1 then the qubit is flipped to $|1\rangle$, if $f_k(x)$ evaluates to 0, the qubit stays at $|0\rangle$. Consequently the qubit simply becomes $|f_k(x)\rangle$. Concluding these calculations, the final equation is Equation 5.2.3.4.2.

$$\frac{1}{2^{n-1}}\left(\sum_{x=0}^{2^n-1}|x\rangle\right)\otimes|f(x)\rangle \qquad \textit{Equation 5.2.3.4.2}$$

Step 3. Now allow the bottom n qubits to decohere and this yields some value, which corresponds either to $f(x_0)$ or to $f(x_0 \oplus a)$. So this sets the top n qubits into a superposition of these two vectors and the state of the computer becomes as in Equation 5.2.3.4.3.

$$\frac{1}{\sqrt{2}}\left(|x_0\rangle + |x_0 \oplus a\rangle\right) \otimes |f(x_0)\rangle \qquad \textit{Equation 5.2.3.4.3}$$

Observe that the Einstein-Podolsky-Rosen paradox was observed here again, the fundament of all quantum computing.

Step 4. Applying the Hadamard transform to the top n qubits results now in Equation 5.2.3.4.4.

$$\frac{1}{\sqrt{2}}\frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}\left((-1)^{x_0 \bullet y} + (-1)^{(x_0 \oplus a)\bullet y}\right)|y\rangle\right) \otimes |f(x_0)\rangle \qquad \textit{Equation 5.2.3.4.4}$$

All vectors y are derived now to two classes. For first class $y \bullet a = 1$. For the second class we have $y \bullet a = 0$. For the first class the Equation 5.2.3.4.5 is obtained:

$$(-1)^{x_0 \bullet y} - (-1)^{x_0 \bullet y} = 0 \text{ for every coefficient.} \qquad \textit{Equation 5.2.3.4.5}$$

Therefore the only vectors **y** that are going to survive this are the vectors perpendicular to vector **a**. The sum evaluates therefore to Equation 5.2.3.4.6.

$$\frac{2}{\sqrt{2}}\frac{1}{2^{n/2}}\left(\sum_{y\bullet a=0}(-1)^{x_0\bullet y}|y\rangle\right)\otimes|f(x_0)\rangle \qquad \textit{Equation 5.2.3.4.6}$$

Measuring the top n qubits returns now always a vector **y** which is perpendicular to **a**. But it can be any vector from the superposition generated by the corresponding measurement on the bottom n qubits. However, if we perform the measurement a sufficient number of times to obtain n different vectors $\mathbf{y}_k$, then we get the set of n independent equations, as in Equation 5.2.3.4.7 below:

$$\begin{aligned} y_1 \bullet a &= 0 \\ y_2 \bullet a &= 0 \\ &\vdots \\ y_n \bullet a &= 0 \end{aligned} \qquad \textit{Equation 5.2.3.4.7}$$

The Equation 5.2.3.4.7 can be solved classically for the vector **a**.

5.3. Grover's Algorithm

5.3.1. Initial Presentations.

In this section the Grover algorithm will be presented in full detail. It is not only a theory, as we know the Grover's algorithm has been successfully realized in NMR

[Chuang98], optical system [Kwiat99] and in cavity QED system [Scully01]. However, all these implementations are restricted to case $N = 4$ for which only one state is required to recover the target state with probability 1. Besides, an extension for greater values of N would be complicated [Ahn00]. We have however simulated Grover for higher values of N using QuiddPro and Matlab.

Let a system have unordered states that are labeled $S_1, S_2, ..., S_N$. Let n be such that $2^n \geq$ N. Let there be an unique state, say S_m , that satisfies the condition $C(S_m) = 1$, whereas for all other states S, $C(S) = 0$ (assume that for any state S, the condition *C(S)* can be evaluated in unit time). The problem is to identify the state S_m. Formulated as this, the Grover's problem is called an "unstructured database search" which name was observed by several authors to be confusing. For an unstructured database, there does not exist any sorting that would aid to select the solution state. The basic idea behind the Grover's algorithm is that one wants to start off with a superposition of all possible database elements. The encoding space for these elements would only be ($\log_2$N) qubits but more importantly a quantum register (or a group of qubits) would be able to hold all possibilities at the same time. This would mean that any operation on the memory would act on all possible elements of the database, in unit time. This is indeed astonishing. The core of the algorithm then revolves around changing the amplitudes vectors (amplitude dictates the probability of each state being observed upon measurement) of the superposed states such that the amplitude vectors of the solutions get magnified at the expense of non solutions. The database metaphor here is not good

for intuitions of an engineer. Let us better assume that we can construct some device, the oracle to tell us if S_m solves the search problem. This device is called historically the Oracle, a logical mechanism device with the ability to recognize solutions to the search problem. Grover implemented this concept using a combination of two transformations performed iteratively for an optimal number of iterations. These are the "selective phase inversion" operation and the "inversion about the mean" operation. Selective inversion of the marked state, followed by the inversion about the mean is also referred to as the Grover Operator, called also the " Grover Loop" and denoted by G. The Grover Operator has the effect of increasing the amplitude of the marked state by $O(1/\sqrt{N})$. Therefore, after $O(\sqrt{N})$ operations, the probability of measuring the marked state approaches the value of "1" [Nielsen00, Grover96].

The definition of the diffusion matrix D: $D_{ii} = 2/N$ and $D_{ij} = -1 + 2/N$ if $i \neq j$ is an inversion of about the average operator. The matrix representations of all operators used are unitary to preserve the normalization constraint. Assume $N = 2^n$ for n input qubits, for simplification.

Here is how the Grover algorithm works; just to explain main idea and in a big simplification.

Step 1. Initialize the quantum memory register to state $|0\cdots0\rangle$

Step 2. Initialize the system to the superposition:

$$\left(\frac{1}{\sqrt{N}},\frac{1}{\sqrt{N}},\frac{1}{\sqrt{N}},.....,\frac{1}{\sqrt{N}}\right)$$ for each N states

i.e. there is the same amplitude $\frac{1}{\sqrt{N}}$ to be in each of the N states of qubits..

Step 3. Repeat the following unitary operations $O(\sqrt{N})$ times.

(a) Let the system be in any state S : change the amplitude a_j to - a_j for S_m such that $C(S_m)=1$, for all other states, leave the amplitude unaltered.

(b) Apply inversion about the average to increase amplitude of S with $C(S_m)=1$. This transformation can be implemented by the diffusion matrix (or diffusion operator) D given above.

Step 4. Measure the first register state which should give us the state S_n where S_n is in $\{0,.....2^n-1\}$. Check $C(S_n)$. This will be the state S_m (i.e. the desired state that satisfies the condition $C(S_m)=1$ with a probability of at least ½).

Step 5. If S_m does not satisfy $C(S_m)=1$, then go to 1. This would be in case the algorithm fails to measure the correct marked state. This is a low probability event, which is however possible.

As we know, placing the register in an equal superposition of all states can be accomplished by applying the Walsh-Hadamard operator [Hayward02]. Using Hilbert space notions, the selective inversion of the marked state in the Grover Operator

means if the system is in any state S and $C(S)=1$, we rotate the phase by π radians, otherwise we leave the system unaltered. This operation can be accomplished with the Oracle as described in [Abe02]. This is also a very natural generalization of the methods from previous sections. In section 5.3.6 we give brief mathematical overview of Grover Algorithm [Chen02] and its Quntumness. We will use the binary string basis in increasing lexicographic order as in Equation 5.3.1.1.

$$|00...00\rangle,|00...01\rangle,|00...010\rangle,...,|11...10\rangle,|11...11\rangle \qquad \textit{Equation 5.3.1.1}$$

for the 2^n dimensional Hilbert space H. In the Figure 5.3.1.1, we utilize the important and elegant results by Barenco et al.[Barenco95] for quantum network representation.

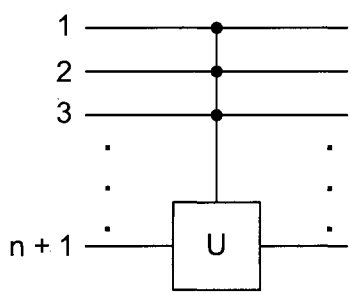


Figure 5.3.1.1: Controlled Quantum gates, the top wire always represents the most significant qubit.

Let $D=\{w_i \mid i=0,1,\cdots,N-1\},(N=2^n)$ be a database oracle which is encoded in an n-qubit quantum computers as $\hat{D}=\{|w_i\rangle \mid i=0,1,\cdots,N-1\}$ with H = span $\hat{D}$. Assume that $|w_0\rangle$ is

the unknown search target in $\hat{D}$. Now, the information through this black box Oracle function is the following $f: \hat{D} \to \{0,1\}, f(|w_i\rangle) = \delta_{i0}$, $i = 0,1, \cdots, N-1$.

Here, we can represent $|w_0\rangle$ as following for the mathematical simplicity and explanation, Equation 5.3.1.2.

$$|w_0\rangle = |a_1 a_2 \cdots a_n\rangle, a_i \in \{0,1\}, \quad i = 0,1,\cdots,n. \qquad \textit{Equation 5.3.1.2}$$

We can write also the Equation 5.3.1.3.

$$|w_0\rangle = \sigma_x^{(i_1)} \sigma_x^{(i_2)} \cdots \sigma_x^{(i_k)} |11\cdots 11\rangle, \qquad \textit{Equation 5.3.1.3}$$

in which

$$\sigma_x^{(j)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad j = i_1, i_2, \cdots, i_k, \qquad \textit{Equation 5.3.1.4}$$

Equation 5.3.1.4 is for Pauli-X rotation matrix (NOT-gate) acting on the j-th qubit. So, $a_j = 0$ for $j = i_1, i_2, \ldots . i_k$ and for all other a_j 's are 1.

Initially we create $|s\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |w_i\rangle$, the uniform superposition of all basis states in Hilbert space H. Now the calculation of matrix I_{w_0} can be obtained as in Equation 5.3.1.5:

$$\begin{aligned} I_{w_0} &= I - \frac{1}{2} \sum_{i=0}^{N-1} \left[|w_i\rangle - (-1)^{f(|w_i\rangle)} |w_i\rangle \right] \cdot \left[\langle w_i| - (-1)^{f(|w_i\rangle)} \langle w_i| \right] \\ &= I - 2|w_0\rangle\langle w_0|, \quad \textit{and} \\ I_s &= I - 2|s\rangle\langle s| \end{aligned} \qquad \textit{Equation 5.3.1.5}$$

Here, both I_{w_0} and I_s are unitary operators. The Grover's unitary operator in the iterative search for $|w_0\rangle$ is G . If initial state is $|s\rangle$ and we apply Grover operator G, k times, we obtain operator $G^k|s\rangle$, $k=\frac{\pi}{4}\sqrt{N}$, in which we will obtain $|w_0\rangle$ with probability close to 1.

5.3.2. Some Insight about Grover ideas: the "Phase Kick-back".

We verified using Matlab and QuiddPro the quantum computational model implemented by the Deutsch-Jozsa Algorithm and the Grover's Algorithm. So we verified our hypothesis how to build the Quantum Oracle for Grover using Quantum Circuits. This is our fundamental idea, from which we can construct all new algorithm for quantum CAD. We call them algorithms but they all use Grover.

From observation, we found that if we do some nice transformation in the input and output state of a Quantum circuit, we can convert the quantum information which is hidden in the phase to the amplitude of the qubit. Here we introduce analysis procedure from input to outputs with Walsh-Hadamard Transform in the input and possible combination of different transforms in the output and keeping our Quantum Oracle in between. By calculating phase in spectral domain we get information which tells us the global issues of the quantum circuit. Besides, we investigated examples of spectral transforms which helped us to explain the general method to create new spectral transforms. The important concept to help understand intuitively the Grover algorithm and similar algorithms is called the "phase kick-back".

Example 5.3.2.1:

This example explains intuitively the concept of "phase kick-back".

As from the Figure 5.3.2.1 we have input vector $|001\rangle$. After Hadamards or other truly quantum gates, in general, the values are complex numbers, and we operate in Hilbert space. This is the hidden information in the quantum state (lost in measurement). As the state vector goes left to right (in quantum evolution or its simulation), each of the complex numbers in vector coordinates will change to another complex number. These vectors can be visualized to help understand the quantum evolution of the circuit. Here we try to develop the kind of an intuition: we have the vector of complex numbers which permanently changes but it preserves all the quantum state vector properties as all the matrices are unitary. If we measure the vector, the sum of squares (sum of probabilities) is equal to one. Hidden information from phase is lost in the measurement, so the information must move by certain transformation from phase to magnitude.

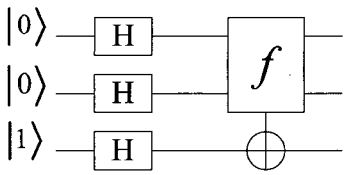


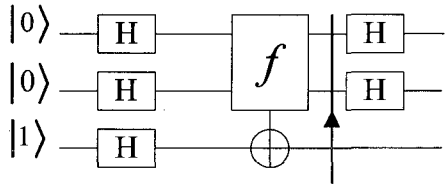
Figure 5.3.2.1: Oracle for function f together with input Hadamards.

Input state to oracle f:

$(|00\rangle + |01\rangle + |10\rangle + |11\rangle)(|0\rangle - |1\rangle)$

Output state of the oracle:

$((-1)^{f(00)}|00\rangle+(-1)^{f(01)}|01\rangle+(-1)^{f(10)}|10\rangle+(-1)^{f(11)}|11\rangle)\otimes(|0\rangle-|1\rangle)$. This is shown in Figure 5.3.2.2.



$$((-1)^{f(00)}|00\rangle+(-1)^{f(01)}|01\rangle+(-1)^{f(10)}|10\rangle+(-1)^{f(11)}|11\rangle)$$

Figure 5.3.2.2: Calculation of the quantum state after oracle. Information is hidden in phase. The Hadamards on data qubits located after oracle transform the phase information to magnitude information.

The state of the inputs $|ij\rangle$ of the oracle that has a output result of 1 is 'tagged' with a negative phase "-1". After Hadamard the solution is "known" in Hilbert space by having value -1. But it is hidded from us. If we observe (i.e. measure) it, we loose it.

The state ψ_{ij} in Hilbert space after oracle may be thus one of the following:

$|\psi_{00}\rangle = -|00\rangle + |01\rangle + |10\rangle + |11\rangle$ if item $|00\rangle$ is data-base is "marked".

$|\psi_{01}\rangle = +|00\rangle - |01\rangle + |10\rangle + |11\rangle$ if item $|01\rangle$ is data-base is "marked".

$|\psi_{10}\rangle = +|00\rangle + |01\rangle - |10\rangle + |11\rangle$ if item $|10\rangle$ is data-base is "marked".

$|\psi_{11}\rangle = +|00\rangle + |01\rangle + |10\rangle - |11\rangle$ if item $|11\rangle$ is data-base is "marked".

Measuring many times will not help as the magnitudes are equal and phases are lost. We need some trick to convert the phase information to one that can be measured. And here comes the great discovery of Deutsch (and Grover) – the Hadamard gates at the output of oracle help (see Figure 5.3.2.2). If we can even slightly change the magnitude we can learn probabilistically the marked states. This is done in Grover loop.

Classically we would need three measurements for two qubits oracles with one minterm marked. But here we need to build extremely complicated quantum circuit after the oracle to convert to magnitude, and next repeat measure and verify using a standard computer until the correct solution is found. Generally in Grover, we see that we have to repeat the Grover Loop $O\sqrt{N}$ times. Now the question is, is this approach practical? Certainly for Database problem it is not practical as the inverted database can be created more efficiently. The reasoning is that if we can build efficient oracle of certain width then basically the length of the oracle is less important, assuming that we have some ways to keep the decoherence fixed. In chapters 12, 13, 14 and 15 I will show problems for which Grover is practical(in future).

We know from Computer Science that every NP hard problem can be reduced in polynomial time to the Satisfiability Problem. SAT is exponential, however in Grover we are improving from N to $\sqrt{N}$, the gain is tremendous, change the exponent to root

of exponent is a big gain. Grover is the most practical quantum algorithm as all Artificial Inteligence problem like satisfiability, graph coloring, Boolean mimization can in principle be reduced to Grover. Grover is a hardware accelerator for any kind of search. In cases that we have backtracking or heuristic strategies, we can further improve our Grover accelerator.

5.3.3. More Ideas on using and Improving the Grover's Algorithm for Quantum CAD Problems.

Quantum algorithms benefit from the superposition principle applied to the internal states of the quantum computer which are considered to be states of a (finite dimensional) Hilbert space. For instance, while classical algorithm needs N steps to search an unstructured database, a quantum Grover algorithm [Grover96] needs only O$\sqrt{}$(N) steps and it can be proved that there is no classical algorithm that would require less steps than O(N) [Zalka99, Boyer96]. Although only few quantum algorithms are now known, many problems can be reduced to some of these algorithms, for instance to Quantum Fast Fourier Transform or to Grover Algorithm. But from this point of view Grover is much better than Shor Algorithm. Thus, any NP-hard problem can be reduced to Grover to give a practically useful and substantial reduction for large values of N, although not as high as in the case of exponential speedup obtained by the famous Shor quantum algorithm [Nielsen00] for integers factorization. The question is: "can we improve Grover Search?"

Simplifying, an oracle is a logic circuit that answers "yes/no" to a question asked to it. Quantum oracle is build from quantum gates to allow superposition and entanglement of its outputs. Remember, that inputs to the oracle are also repeated as some of its outputs and they encode the solution using the "phase kick-back" [Nielsen00]. Without going yet to full details how an oracle works in a quantum algorithm, let us observe here only that the oracle must be built from truly quantum gates and that many oracles include arithmetic, logic, and mixed blocks. If one only knows how to build a respective oracle, Grover algorithm and its modifications would be immediately useful to solve many problems when the physical quantum computers will become available. It is therefore important to study methods and algorithms to build various types of oracles (next chapters). The problem of building various classes of oracles or their blocks (components) is well known in case of binary quantum circuits (see [Nielsen00] and recent review about automatic synthesis in [Perkowski04]). Many papers how to synthesize them from binary quantum gates, or proposing general-purpose logic synthesis algorithms for binary quantum circuits have been recently published, which can be used together with the methods derived in Chapters 7 – 15 of this thesis.

We know that orthogonal transformations can be used to transform a Boolean function into its unique representation in the spectral domain. Many such transforms are surveyed by Hurst et. al.[Hurst85]. In particular, the Hadamard transform is susceptible for computing purposes. Each coefficient of Hadamard transform gives

some global information about the function. The indices of a coefficient represent which input variables the coefficient correlates. In the general case , for a given ***F***, there is one **zeroth order coefficient.** This term reflects the correlation of ***F*** to a constant value. The orders of coefficients increase upto **nth order**. The higher order coefficients correlate to the **exclusive or** of all the input variables specified in the coefficient index. Manipulations between spectral and Boolean domains are easy since forward and inverse operations involve applications of the same transform. Walsh-Hadamard transform bases are waves and the Walsh coefficients correspond to "chess patterns" in KMaps. They are thus used in communication, encryption, image processing and logic synthesis.

These properties of Hadamard Transform are used in Deutch, Deutch-Jozsa, Berstein-Vazirani and Simon algorithms. They should be also used in Grover-based problem solving, but this subject is absent from literature.

Concluding, the Grover algorithm in CAD applications can be improved by:

1) Using special cases and related heuristics.
2) Using parallelism on the level of quantum process, i.e., many quantum computers working in parallel (as in ensemble quantum computing, or in other parallel computing).
3) Using spectral transforms in synthesis, i.e., the Quantum Fast Fourier Transform. Hadamard Transform, Reed-Muller Transform, etc.

4) Using parallel quantum accelerators working with standard computers that control, verify and interpret data from quantum computers.
5) Reconfiguring quantum hardware dynamically.
6) Using phase kick-back and similar tricks.

Although Grover can not be improved as a general search algorithm, it can be improved for particular problem instances.

5.3.4. Calculations and Experimental Results.

We calculated (simulated) several circuits. We calculated the Deutsch algorithm and also Grover algorithm by using QuiDDPro Simulator. We build the Oracle in QuDDPro and also showed the visualization of quantum evolution of quantum circuits. Every quantum algorithm is basically Oracle plus spectral transform on subsets of inputs and outputs. We illustrated in simulations the trick of putting the information in phase and the quantum parallelism. We found experimentally that inputs to the oracle, repeated as some of its outputs, encode the solution using the so-called "phase kick-back" [Brassard97]. Every NP complete problem can be reduced to Satisfiability, Graph Coloring or similar problem. Thus our simulation results confirm the validity of our novel method to build a "general purpose" Quantum CAD design system (chapters 7 – 15).

In several architectures, we use Grover Algorithm to evaluate the condition that number of ones in the co-domain of certain mapping is less than certain user-selected threshold value (graph coloring – chapter 13).

A block diagram of our simulated version of quantum circuit of Grover Algorithm is shown in Figure 5.3.4.1.

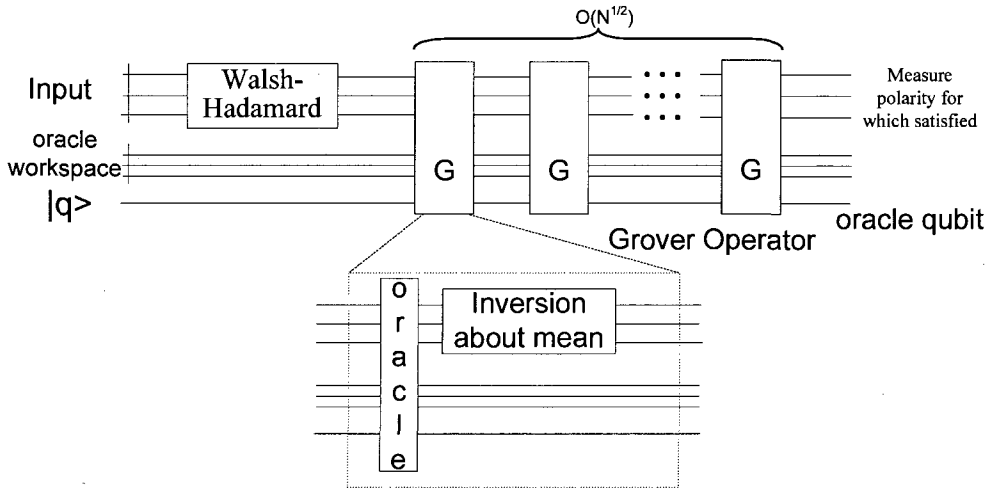


Figure 5.3.4.1: Grover Algorithm Block Diagram.

Circuit design is described in chapters 7, 8, 9. Blocks are presented in chapter 11. Oracles are in chapters 12, 13, 14 and 15. Inversion about the mean will be discussed in full detail in the remaining of this chapter. The discussion of more detailed physical aspects of the implementation of Grover algorithm is beyond the scope of this dissertation.

5.3.5. The Detailed Layout of the Grover Algorithm.

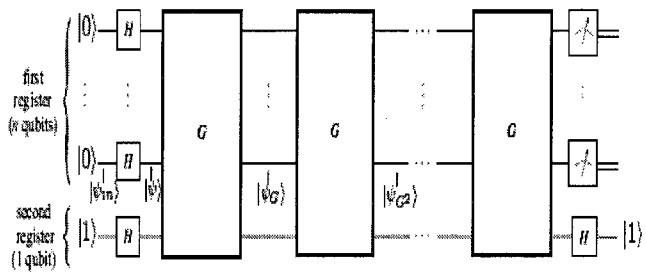


Figure 5.3.5.1: The Grover Algorithm block diagram. Here, the G's in the boxes represent Grover operators as in Figure 5.3.4.1.

The Hadamard gates that act upon all inputs make every element in the state vector equal. This state vector is represented as state $|\varphi\rangle$ in Figure 5.3.5.1. The mathematical expression of $|\varphi\rangle$ (the initial state) is shown in Figure 5.3.5.2.

$$
\begin{aligned}
&= H^{\otimes n}|0,...,0\rangle \\
&= (H|0\rangle)^{\otimes n} \\
&= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes n} \\
&= \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle .
\end{aligned}
$$

Figure 5.3.5.2: the mathematical representation of initial state $|\varphi\rangle$, first register(see Figure 5.3.5.1).

The second input register is a bit 1, and serves to make the G-iteration work. After going through a Hadamard gate, it assumes the output state of the Hadamard gate (see Figure 5.3.6.1), and is denoted by state $|-\rangle$. The inputs go through a number of G-iterations, after which the state vector will become the solution state.

5.3.6. The G-iteration.

Grover's Algorithm transforms the basic state $|\varphi\rangle$, where the probabilities of measuring each state are equal, into a state that has an overwhelming probability of measuring the solution. This transformation is achieved through G-iterations, illustrated in Figure 5.3.6.1.

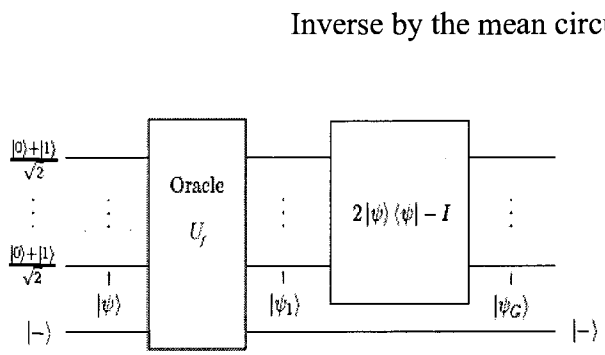


Figure 5.3.6.1: The first G-iteration

The G-iteration is composed of the Oracle U_f and the "inverse by the mean" function $2|\varphi\rangle\langle\varphi| - I$. Grover's Algorithm is meant to significantly increase the amplitude of the desired element. In order to be able to transform the element state's amplitude, we

need a function that will specifically act on it, which will be the basis of all later transformations of the element. This is *f(i)*, which is illustrated in Figure 5.3.6.2.

$$f(i) = \begin{cases} 1 \text{ if } i \text{ is the searched element } (i_0) \\ 0 \text{ otherwise.} \end{cases}$$

Figure 5.3.6.2: Function f(i)

U_f is an oracle that is based on the function f(i). U_f is designed to conduct a phase shift on the desired element, i_0, which corresponds to it gaining a negative phase in the state vector. This does not affect the other inputs at all; as probability amplitudes are squared, negative amplitude makes no difference. The function of U_f is illustrated in Figure 5.3.6.3.

$$\begin{aligned} U_f\left(|i\rangle\,|-\rangle\right) &= \frac{U_f\left(|i\rangle\,|0\rangle\right) - U_f\left(|i\rangle\,|1\rangle\right)}{\sqrt{2}} \\ &= \frac{|i\rangle\,|f(i)\rangle - |i\rangle\,|1 \oplus f(i)\rangle}{\sqrt{2}} \\ &= (-1)^{f(i)}\,|i\rangle\,|-\rangle\,. \end{aligned}$$

Figure 5.3.6.3: Function U_f

This U_f marks the i_0 with a minus sign, but the amplitude is yet to be increased. The state vector after U_f matrix can be considered $|\varphi_1\rangle$, i_0 is increased by the second part of the G-iteration the "inverse by the mean" circuit described by the unitary matrix

$2|\varphi\rangle\langle\varphi| - I$. The operation of unitary matrix $2|\varphi\rangle\langle\varphi| - I$ cannot easily be explained in terms that are not geometric (Figure 5.3.6.4).

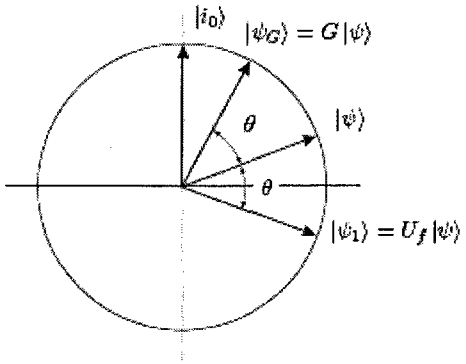


Figure 5.3.6.4: Geometric representation of G-iteration.

$2|\varphi\rangle\langle\varphi| - I$ is the operator that occurs between φ_1 and φ_G. It flips the state over the initial state $|\varphi\rangle$. This increases the amplitude of i_0. $|\varphi\rangle\langle\varphi|$ when applied to any state, brings up the initial state $|\varphi\rangle$. The operators U_f and $2|\varphi\rangle\langle\varphi| - I$ combine to increase the amplitude of the desired state i_0 by an arbitrary amount theta, which is smaller when the Grover's "database" is larger (thus requiring more G-iterations).

It is important to note that Grover Algorithm is usually discussed as a "stand-alone" quantum algorithm executed on a single quantum computer. However, a more interesting approach is to consider a parallel system of computers, each of them being a classical computer with its reconfigurable quantum accelerator processor that can be dynamically reprogrammed and thus adapt to any given particular problem. This

approach will be presented in chapter 6. In chapters 12, 13, 14 and 15, I will discuss various applications of Grover for which I constructed oracles.

5.4. The Matlab Simulations.

5.4.1. The need for a simulation

We need to prove that our Oracle complies with Grover's Algorithm. If it did not, then we would have to find an entirely different model for our algorithm, as our existing one would be faulty. The simulations were useful exercises as we often found several errors in our design and text files. I used QuiDDPro and MATLAB simulators to explore and verify my idea of designing Grover Oracle for various problems and their versions discussed in chapters 12 and 13.

5.4.2. The method of simulation

We used the MATLAB program to simulate the circuit, as we have no quantum computers to do so. MATLAB uses matrices, so we would have to derive the operation matrix of our algorithm, and the test it on the initial state vector (see Figure 5.4.2.2). We used a simplified graph (Figure 5.4.2.1), and a simplified version of the Oracle, consisting only of the Graphic rule checker (Figure 5.4.2.2). The entire final circuit of the oracle is shown in Figure 5.4.2.3.

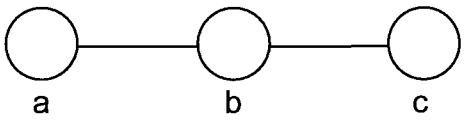
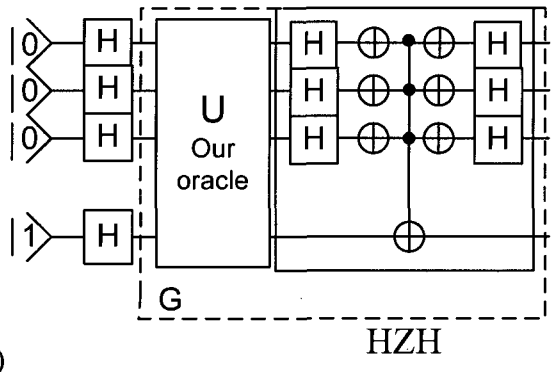
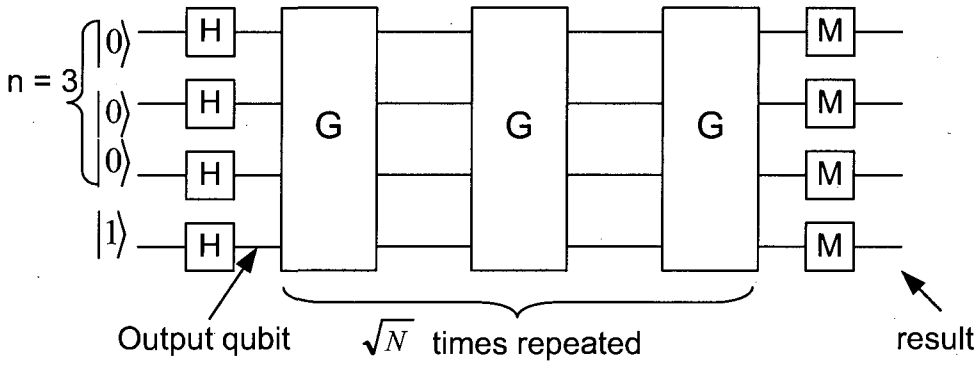


Figure 5.4.2.1: The graph with 3 nodes for coloring to be simulated.



(a)

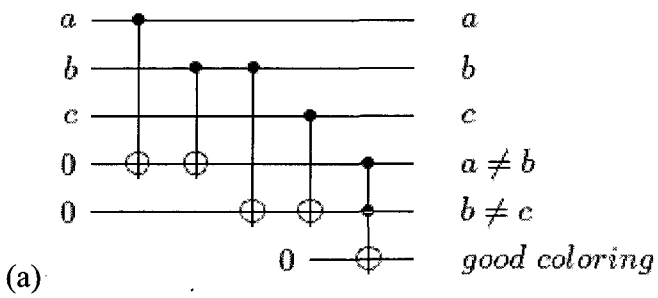


(b)

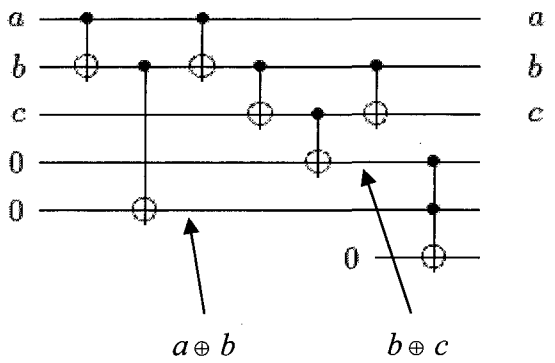
Figure 5.4.2.2: The Grover Loop for graph coloring of the simple map (planar graph) from Figure 5.4.2.1. (a) Gives the complete "Grover Loop" circuit, denoted by G. (b)This circuit should be iterated $\sqrt{N} = \sqrt{2^n} = \sqrt{2^3} = 2\sqrt{2} = 2.1.41 \approx 3$ *times.*

We find the matrix of this Oracle below. By creating the matrices of U1, U2, etc., we can find the total matrix of the comparator, which is denoted as Mcomp here. Our initial naïve design of the oracle is shown in Figure 5.4.2.3a. Next I improved the design to the circuit from Figure 5.4.2.3b and finally I got the idea of the circuit from

Figure 5.4.2.4b. This circuit was so simple and beautiful that it actually made me think about the role of CNOT gate which ultimately led to the invention of Affine gates.



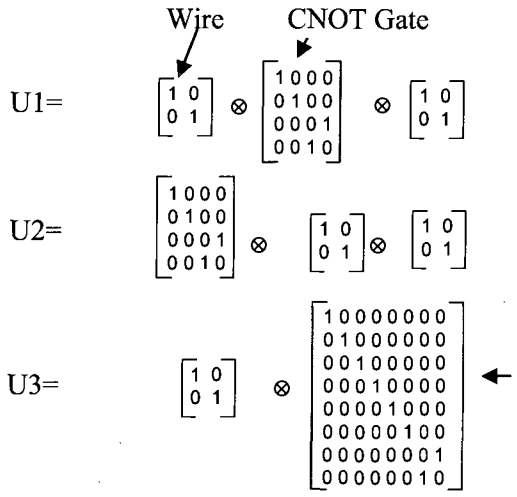
(a)



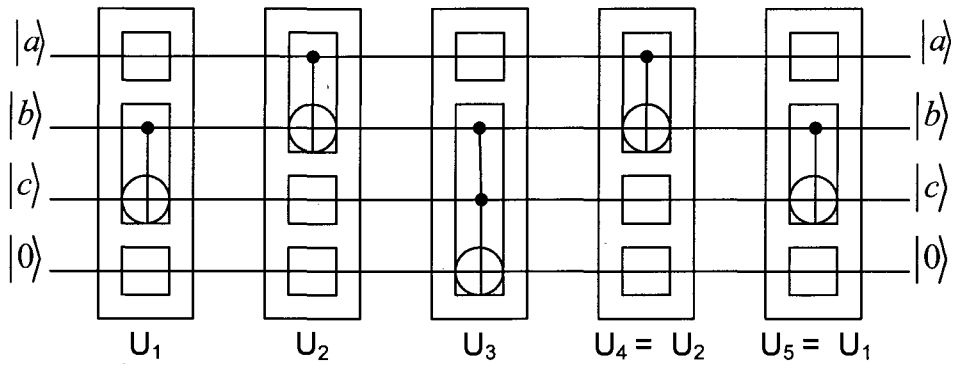
(b)

Figure 5.4.2.3: The Graph Coloring checking oracle for the graph from Figure 5.4.2.1. (a) the circuit created directly from problem definition and without any optimization, (b)the next variant of the circuit uses mirrors for $a \oplus b$ and $b \oplus c$ and is more expensive than the circuit from Figure 5.4.2.1a.

This circuit concept leads however to the circuit with one less ancilla bit (Figure 5.4.2.4) which simplifies much the Matlab simulation.



(U4 is identical to U2, U5 is identical to U1)
The total U matrix of the Oracle is
U=U5 * U4 * U3 * U2 * U1
(a)



(b)

Figure 5.4.2.4: Calculations of the Unitary (permutative) matrices from the oracle. (a) The calculations of the matrices, (b) the complete oracle circuit with partitioning to matrices from Figure 5.4.2.4a.

The HZH is the Zero Shift subcircuit in the Grover's Algorithm (Figure 5.4.2.2). The circuit analyzed in Figure 5.4.2.5 circuit is composed of basic gates. These gates have respective matrices. "H" is the matrix of a Hadamard, "Minv" an inverter, "wire" a wire, and "Toffoli 3" a 3-input Toffoli gate. Using Kronecker multiplication, we can

find the matrices of each column. The columns of HZH are denoted as HN, where N is a number. N starts from the rightmost column, and increases as you go to the left. These column-matrices are multiplied together to create the total matrix of the circuit. This is done below. Mh is the initial Hadamard transform of the circuit, and U is the matrix of the Oracle. Vinit is the initial input vector.

H1= H⊗H⊗H⊗wire
H2=Minv⊗Minv⊗Minv⊗Wire
H3=(Toffoli 3)

(H4 is equal to H2, H5
equal to H1)

The total matrix HZH is equal to:
H1 * H2 * H3 * H2 * H1

So, the total matrix for one iteration is:
(HZH * U * Mh)Vinit

Figure 5.4.2.5: Analysis of the single iteration of Grover Loop.

MATLAB calculated all these results plus the entire Unitary matrix of the Grover Algorithm for this case. The results were consistent with what would happen if truly quantum Grover's Algorithm was applied to the problem. Thus, we prove that our Oracle can work with the truly quantum Grover's Algorithm to solve a very simple graph coloring problem.

5.5. Conclusion

The oracle design for Grover Algorithm has been successful in creating an Oracle for graph coloring, as verified by several MATLAB simulations. The concepts presented in this thesis were learned alongside the creation of the project. Important concepts such as logic synthesis and simulation were necessary for the oracle design theory, and their usage solidified my knowledge of them. This oracle design was my first application of quantum computing, and so proved invaluable to my education of all presented concepts. This led also to invention of new gates and oracles, as well as new blocks. For instance the circuit from Figure 5.4.2.4 attracted me to the idea of affine Toffoli gates. The powerful idea that influenced me comes from Raymon Lullus (XIII century) who influenced Descartes who influenced in turn George Boole. Boole said "every logic problem can be formulated as a logic equation". We call them now the "Boolean Equations" to honor George Boole and the name of Raymon Lullus is unfortunately forgotten.

CHAPTER 6

Tree Search, Parallel Search and Quantum Parallel Search

6.1. Introduction. The essence of parallel quantum search.

This chapter contains a description of our new approach to problem-solving and learning. The method presented in this chapter is an improved search method that applies both to classical and quantum search.

There are several approaches to find solutions in combinational problems. One group of approaches are based on tree searching. Algorithms such as depth-first-search, breadth-first-search, tabu search or A* search are used. Another approach uses Genetic Programming or Genetic Algorithm. Yet another approach uses Simulated Annealing. Learning can be incorporated in one way or another into any of these algorithms.

Here we will show a new approach where several algorithms are combined together and that is specialized for minimization of multi-level, binary and multiple-valued logic networks from various types of gates, in particular in AND/EXOR, Galois and Linearly Independent Logic.

Search can be realized on a serial (one-processor computer) and on a parallel computer. Parallelism gives of course the increase of the processing power thanks to many processors working in parallel that can be used to decrease the processing time.

There is however also another advantage of parallelism. It is that the processors can use different algorithms and thus one of them can find some coefficients or bounds that can be next used by all processors as the bound constraining their search thus improving the total processing time to find the solution. Also, using FPGAs or other reconfigurable system allows to use the learned problem characteristics to modify the structure of the computing systems or/and its processing units. All these ideas can be applied to the quantum search as well as to the standard search.

We can characterize a general parallel quantum search method as having the following properties:

1. The general search method uses "unit processors" to perform "canonical searches".
2. Each canonical search uses certain strategy, this strategy is quantum or not. Each canonical search searches certain subspace of the entire problem space.
3. Each canonical search searches certain tree in a complete or incomplete search manner. Its work can be stopped by other processor to load a new search problem to its processor or the search parameters can be updated.
4. There are three types of parallelism related to quantum computing:
    1) The quantum algorithm as introduced in Chapter 5 of this thesis. We will call it the "standard quantum computer". This computer has quantum

parallelism as represented in Grover algorithm. This parallelism is based on superposition and entanglement and was fully explained in chapter 5.

2) In contrast to "standard quantum computer" the quantum computer can be "ensamble quantum computer." In ensamble quantum computer many standard quantum computers work in parallel as one unit, performing exactly the same algorithm. Observe that if a standard quantum computer is for instance in state V_0 then, when measured, it gives probabilistically 0 or 1 with probability 1/2 each. We thus do not know if the computer was in state 0 or V_0. Quantum Ensamble computer however works differently. Let us assume that 10,000 standard quantum computers in this ensamble computer are in state V_0, then during the measurement statistically 5,000 computers will read 0 and another 5,000 computers will read 1. Thus we statistically know that the state of each computer was on a big circle (Equator) of the Bloch sphere. We have thus more information than in the case of a standard quantum computer and we can distinquish state 0, 1 and V_0 in this single (ensamble) measurement.

3) Finally, there can be a set of standard quantum computers or a set of ensamble quantum computers that work in parallel. This is similar to a parallel computing system of normal computers, and various structure and network types of parallel computers can be used. This is the most general model of computing of this thesis.

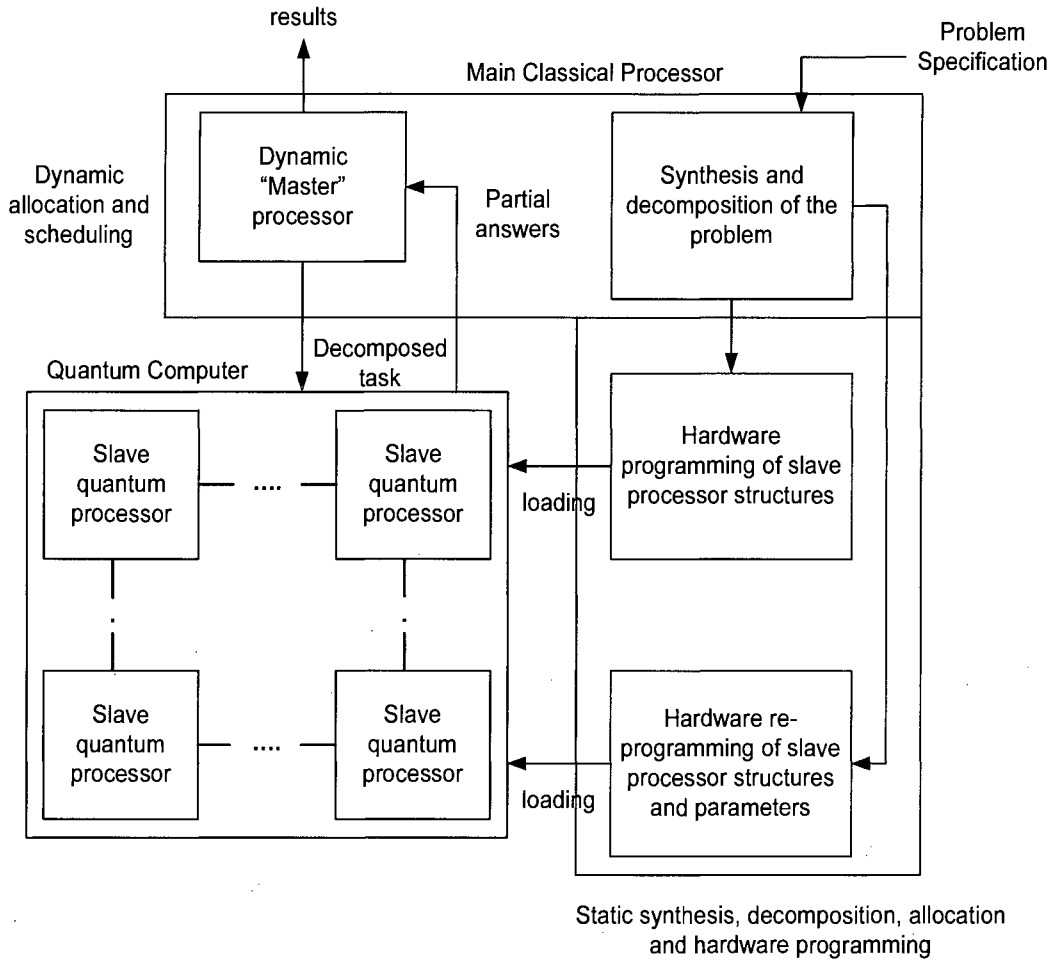


Figure 6.1.1: Hierarchical control Figure.

The entire search can be thus decomposed to many quantum and standard computers. When we talk in this dissertation about a search tree, one has to understand that various parts of this tree can be expanded separately in various processors and by various algorithms and computational mechanisms 1), 2), 3) as above. The schematic diagram of a parallel quantum computer is presented in Figure 6.1.1.

6.2. Advanced Search Method

6.2.1. Introduction to Advanced Search Methods

One of the most important components to create successful programs for many CAD applications is developing a good search strategy that is based on the particular problem to be solved and its problem-specific heuristics.

In principle, better search methods either use some kind of heuristics, or utilize some *systematic* search strategies that *guarantee*, at least local, optima. One convenient and popular way of describing such strategies is to use the concepts of *tree searching*. Tree is a structure of sub-trees, these subtrees can be be searched in parallel or in series. Each subsearch can be executed on a standard computer, or a parallel or a quantum computer. The theory that we present here relates thus both to the entire tree search problem and to each subsearch problem.

The problem of creating complex heuristic strategies to deal with combinatorial problems in CAD is very similar to that of general problem-solving methods in Artificial Intelligence and Robotics. There are five main principles of problem solving in **AI**:

- state-space approaches including constraint satisfaction,
- problem decomposition,
- automatic theorem proving,

- rule-based systems,
- learning methods (neural nets, abductive nets, immunological, fuzzy logic, genetic algorithm, genetic programming, Artificial Life, etc.).

Since we will limit the discussion in this chapter to the description of the state-space principle, the approach that we will use is based on the assumption that any (combinatorial) problem of our class can be solved by searching some space of states. The space is enormously large in practical problems and it has a certain structure or not. If the space has no structure, not much can be done other than making the search as parallel as possible. But usually the space has some type of structure and this structure should be used to design the improved search method.

Search in space of states seems to be the best approach because of its simplicity, generality, adaptability, parallelization, parameterization and other nice properties. By using this approach, the sets of problems within this framework are not greatly restricted.

There are also other reasons for choosing the state-space heuristic programming approach:

6.2.1a. The combinatorial problem can be often reduced to integer programming, dynamic programming, or graph-theoretic problems. The graph-theoretic approaches include in particular, the set covering, the maximum clique, and the graph coloring. The computer programs that would result from pure,

classical formulations in these approaches would not sufficiently take into account the specific features and heuristics of the problems. Instead reducing to known models, we will create our own general model, and "personalize" it to our problems. For instance, instead of using a standard (proper) graph coloring approach, we may formulate the compatible graph coloring problem, an adaptation of proper graph coloring that uses also other constraints. Moreover, we use heuristic directives based on our data to solve the modified/adapted problem efficiently. The problems are rather difficult to describe using these standard formulations. The transition from the problem formulation, in these cases, to the working version of the problem-solving program is usually not direct and cannot be automated well. It is difficult to experiment with strategy changes, heuristics, etc. These parameterized experimentations are one of our main goals here in case of standard computers. The same rules and methods can be however used also in future to quantum computers. We aim at the model's flexibility, and of the model's being able to easily tune its parameters experimentally. In a sense, we are looking at a "fast prototyping" possibility. Now we cannot use our model fully on quantum simulators, as we do not dispose a parallel system of quantum simulators.

6.2.1b. Some of these combinatorial problems (or similar problems) have been successfully solved using the state-space heuristic programming methods. The state-space methods include some methods that result from other **AI**

approaches mentioned above. Some backtracking methods of integer programming, and graph-traversing programs used in graph partitioning and clustering methods, are for instance somewhat similar to the variable partitioning problem. They can be considered as special cases of the problem-solving strategies in the space of states.

6.2.1c. Other problems were solved using Genetic Algorithm as it was not possible to use another type of search because of problem size. Hopefully, quantum computing will allow to create algorithms of higher quality and efficiency, including exact minimizations for problem instances that are not possible on standard computers.

6.2.1d. We found that there are, in most cases, some straightforward procedures to convert search algorithms to quantum oracle problem formulations. This is only a beginning of research and we are mostly restricted to Grover Algorithm.

Roughly speaking, several partial problems in logic CAD can be reduced to the following general model:

6.2.2a. The rules governing the generation of some set S, called *state-space*, are given. This space can be created in series and in parallel, in standard world or in quantum world. This set is in most cases implicitly defined, not explicitely. Explicit formulation is only in the simplest games and puzzles used as illustrations.

6.2.2b. *Constraint conditions* exist which, if not met, would cause some set $S' \in S$ to be deleted from the set of potential solutions. Again, this deletion can be done in series or in parallel, in standard or in quantum computing spaces.

6.2.2c. The *solution* is an element of S that meets all the *problem conditions*.

6.2.2d. The *cost function* F is defined for all solutions. This function is calculated in series, in parallel or in a mixed serial/parallel way. It is calculated by software or by hardware. The hardware oracle block is combinational in quantum synthesis but it may have memory and automata components in quantum or non-quantum hardware. We are not interested in quantum automata in this dissertation.

6.2.2e. The solution (one, more than one, or all of them) should be found such that the value of the cost function is optimal *(quasi-optimal)* out of all the solutions.

A problem condition *pc* is a function with arguments in S and values *true* and *false*. For instance, if set S is the set of natural numbers:

$$pc_1(x) = true - if\ x\ is\ a\ prime\ number;\ false\ -\ otherwise$$

In general, a *problem* can be defined as an ordered triple: $P = (S, PC, F)$, where:

6.2.3a. PC is a set of predicates on the elements S of, called *problem conditions*. In standard design the conditions are checked for one candidate at a time.

However, the power of quantum computing is that all conditions are verified for all the states being solution candidates in parallel.

6.2.3b. *F* is the cost function that evaluates numerically the solutions. Solution is an element of S that satisfies all the conditions in *PC*.

The *tree search method* includes:

6.2.3b.1. The problem *P*,

6.2.3b.2. The constraint conditions,

6.2.3b.3. *Additional solution conditions* that are checked together with the problem conditions,

6.2.3b.4. The *generator of the tree*. Generation can be done in parallel, in series, in quantum, in standard software, using sequential or combinational circuit.

6.2.3b.5. The *tree-searching strategy*. The strategy can be parallel, serial, quantum, standard software, etc. As discussed earlier. The strategy is usually composed of several sub-strategies. Only in didactic examples we will use pure strategies that are not mixed.

Additional solution conditions are defined to increase the search efficiency.

For instance, assume that there exists an auxiliary condition that is always satisfied when the solution conditions are satisfied, but the auxiliary condition can be tested less expensively than the original solution conditions. In such case the search efficiency is increased by excluding the candidates for solutions that do not satisfy this auxiliary condition. This can be done in the same search process or in another search

process, executed subsequently. Standard processor gives more flexibility but quantum computer gives more processing power and parallelism.

The additional solution conditions together with the problem conditions are called *solution conditions*. The method is *complete* if it searches the entire state-space and thus assures the optimality of the solutions. Otherwise, the entire space is not searched and the search methods will be referred to as *incomplete methods*. Obviously, for practical examples most of our searches will use incomplete search methods. Although quantum computer gives enormously high processing power comparing to standard computers, they will be also restricted as we will formulate more complex problems for them. Thus incomplete and approximate methods will be always of use, only the complexity of the problems will dramatically increase.

We will illustrate these ideas for the case of the minimal covering (set covering, unate covering) problem, which has several applications. For instance, the problem is defined as follows:

A. The problem is represented as a rectangular table with rows and columns. Each column is to be covered with the minimum total cost of rows. The state-space S is a set that includes all of the subsets of the set of rows of the covering table (rows correspond for instance to prime implicants contained in a Boolean function [Kohavi70].

B. The solution is an element of S that covers all the columns of the function.

C. A cost function assigns the cost to each solution. The cost of a solution is the number of selected rows. It may also be the total sum of the selected rows and their costs.

D. A solution (set of rows) should be found that is a solution and minimizes the cost function.

E. Additional quality functions are also defined that evaluate states and rows in the search process.

F. This process consists of successively selecting "good" rows (based on the value of the quality function), deleting other rows that cover fewer of the matrix columns (these are the *dominated rows*), and calculating the value of the cost function.

G. The cost value of each solution cover found can then be used to limit the search by *backtracking*.

H. This process can be viewed as a search for sets of rows in the state-space, and can be described as a generation of a tree*(solution tree)* using rows as *operators*, sets of rows as *nodes of the tree*, and solutions as *terminal nodes*.

A combinatorial problem of a set covering type can either be reduced to a covering table, or solved using its original data structures. Finally it can be reduced to a logic equation (Petrick Function) which is evaluated in software, in standard (classical oracle) or in a quantum oracle. It has been shown by many authors [Cordone01], that

the following classical logic synthesis problems, (among many other), can be reduced to the Set Covering Problem.

These problems are:

(1) the PLA minimization problem.

(2) the finding of vacuous variables.

(3) the column minimization problem.

(4) the microcode optimization problem.

(5) the data path allocation problem.

(6) the Three Level AND/NOT Network with True Inputs (TANT) minimization problem.

(7) the factorization problem.

(8) the test minimization problem, and many other classical logic synthesis problems.

(9) the layout minimization problems, including ancilla bits minimization in quantum circuits.

(10) the ESOP minimization problem.

Therefore, the Set Covering, Even/odd covering, Binate covering, and many similar (selection) problems can be treated as a *generic logic synthesis subroutine*. Several

efficient algorithms for this problem have been created [Dill97, Perkowski99, Files97, Files98, Files98a]. Some of these algorithms can be used also to create oracles.

The methods presented here can be applied to all problems presented in chapters 2, 3, 4, 7 – 11 and specifically to:

1. Finding minimum realization in the sense of number of elementary pulses for quantum gates (chapter 2)
2. Finding minimum realization of PPRM for incompletely specified function
3. Finding minimum realization of FPRM for completely and incompletely specified function
4. Finding minimum realization GRM for completely and incompletely specified functions
5. Finding minimum realization for all kinds of affine circuits for various polarities.
6. Finding minimum realizations for all other canonical forms and ESOP.

We can use the search ideas from this chapter to solve efficiently all these problems. Some will be illustrated. Equivalently, I believe that some of the ideas from the literature about optimization and oracle construction can also be used to extend the search framework presented by us, both its classical and quantum aspects.

Moreover, various methods of reducing a given problem to the Set Covering Problem exist. These methods would result in various sizes of the covering problem. By a smart approach, the problem may still be NP-hard, but of a smaller dimension. For a particular problem then, one reduction will make the problem practically manageable, while the other reduction will create a non-manageable problem. This is true, for instance, when the PLA minimization problem is reduced to the set covering with the signature cubes [Brayton87] as columns of the covering table, rather than the minterms as the columns of this table. Such reduction reduces significantly the size of the covering table. Similar properties exist for the Graph Coloring, Maximum Clique, reversible logic synthesis, ESOP minimization, quantum circuit minimization and other combinatorial problems of our interest. Although the problems are still NP-hard as a class, good heuristics can solve a high percent of real life problems efficiently. This is because of the Occam's Razor principle – circuits described by engineers are not random circuits – the random circuits are the most difficult to minimize, but hopefully there is no use to minimize them so they will be not a subject of optimizations.

Many other partial problems for CAD of classical computers, including those in high-level synthesis, logic synthesis, and physical CAD, can also be reduced to a class of NP-hard combinatorial problems that can be characterized as the *constrained logic optimization problems*. This is a subclass of the constraint satisfaction problems.

These problems are described using binary and multiple-valued Boolean functions, various graphs and multi-graphs, arrays of symbols or other specifications. Some constraints are formulated on these data, and some transformations are executed in order to minimize the values of cost functions. These problems include Boolean satisfiability, tautology, complementation, set covering [Hochbaum82], clique partitioning [Pozak95], maximum clique [Jou93], generalized clique partition, graph coloring, maximum independent set, set partitioning, matching, variable partitioning, linear and quadratic assignment, encoding, and others. These entire problems can be realized as quantum oracles, and we will illustrate several of them in chapters 12, 13, 14 and 15.

With respect to high importance of these problems, several different approaches have been proposed in the literature to solve them. These approaches include:

1. Mathematical analyses of the problems are performed in order to find the most efficient algorithms (the algorithms may be exact or approximate). If this cannot be achieved, the algorithms for particular sub-classes of these problems are created. This can speed up solving problems on large classes of practical data, in spite of the fact that the problems are NP-hard so that no efficient (polynomial) algorithms exist for them. For instance, the proper graph coloring problem is NP-hard, but for a non-cyclic graph there exists a polynomial complexity algorithm. How practical is the polynomial algorithm, it depends only on how often non-cyclic graphs are found in any given area of application where the graph coloring is used.

2. Special hardware accelerators are designed to speed-up the most executed or the slowest operations on standard types of data used in the algorithms.

3. General purpose parallel computers, like message-passing hypercube processors, SIMD arrays, data flow computers and shared memory computers are used [Duncan90]. Some ideas of parallel, systolic, cellular and pipelined hardware can be applied to building quantum oracles. For instance, the sorter absorber circuit that I use for converting lists to sets in quantum oracles (chapter 13) has been adapted from pipelined sorters used in standard hardware.

4. The ideas of Artificial Intelligence, computer learning, genetic algorithms, and neural networks are used, also mimicking humans that solve these problems. In this dissertation we also follow some of these ideas [Nilsson71].

6.3. Multi-strategic Combinatorial Problem Solving

6.3.1. Basic Ideas of Multi-strategic search

The goal of this section is to explain how the general objectives outlined in sections 6.1 and 6.2 can be realized in programs and hardware systems to solve combinatorial problems. It is well-known that the difference between hardware and software has been recently blurred with the introduction of reconfigurable computers and Field Programmable Gate Arrays. It should be thus clear to the reader that many of ideas

that we present below are applicable to both software and hardware, including quantum oracles.

Some of the methods presented here have been already programmed, some other not yet. Some have been used to design quantum oracles from next chapters, some other are not incorporated into the thesis as they lead to very complex circuits. I am afraid that they would expand the thesis too much. Our interest is in a uniform explanation and the creation of state-space tree search methods that would be general and independent on the computing substrate. Our first goal is *Fast Prototyping*. By fast prototyping, we want the program to be written or a system to be designed in such a way that the developer will be able to easily change the program/hardware for each experiment. This is illustrated by the set covering software and by the way of building respective logic oracles in chapter 12.

Our general methodology includes an important component of changing the problem description variants and create various search strategies for different tree search methods to optimize the efficiency of the search.

The tree-search strategy was created by selecting respective classes and values of *strategy parameters*. The creation of multiple variants of a tree-searching program, that could require weeks of writing and debugging code would then be possible in a

shorter period of time. Some efficiency during execution will be lost, but the gain of being able to test many variants of the algorithm will be much more substantial. The *behavior* of the variants of the tree search methods will then be compared and evaluated by the developer to create even more efficient algorithms.

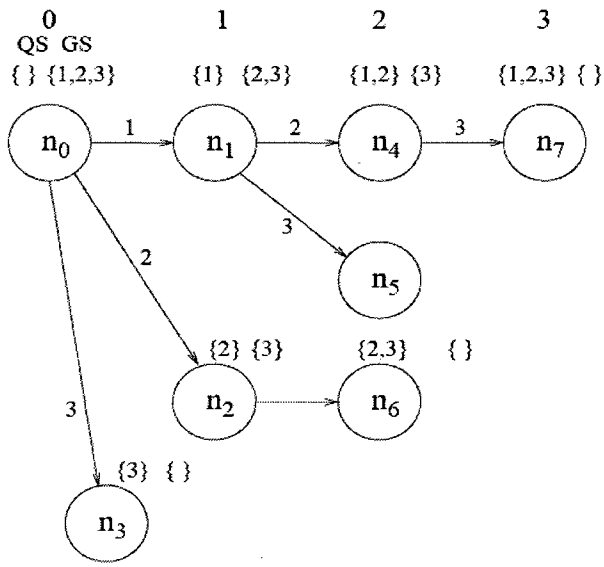


Figure 6.3.1.1: Example of T_1 type tree generator of a full tree.

Figure 6.3.1.1 presents a tree generator. Such generator can be used in software or standard hardware. It generates all subsets of a set of elements {1, 2, 3}. This generation can be done in series or in parallel. It can be decomposed to many subtrees. In case of quantum processing, the generation is done as creating binary vectors corresponding to subsets and all these vectors are superposed within a single unit search subprocess. For instance, we can imagine a hierarchical system that has parallel structure of quantum computers. The initial problem for set {1, 2, 3} is created in a

processor corresponding to node n_0. It is decomposed serially to two sub-problems, Sub-problem-1 is for the sub-tree with nodes n_1, n_4, n_7, n_5. Another sub-problem, Sub-problem-2 has nodes n_2, n_6 and n_3. Observe that the Sub-problem-2 is the complete search of all subsets of set {2, 3} so it has a general nature which can be used to build a quantum computer for all subsets of set {2, 3}. The Sub-problem-1 includes all solutions with element 1, and in addition it searches the subsets of set {2, 3}. Thus another quantum computer can be constructed for set {2,3} which in addition knows that element 1 is selected. These quantum computers can be realized dynamically using a quantum software/hardware design approach that extends standard FPGAs. We call it Reconfigurable Quantum FPGA. In this simple example we have Processor-0 which is a standard processor, and two subordinated to it processors: Processor-1 and Processor-2 that execute Sub-problem-1 and Sub-problem-2, respectively. Observe that when one of the quantum processors finds a solution it informs the Processor-0 about the cost value and the Processor-0 can change its strategy of giving values and sub-problems to subordinated quantum processors. It can also reconfigure them, by practically designing them from scratch using quantum circuit design. For instance, in case of graph coloring, if a proper coloring with a new cost value k is found, if this value is much lower than the current assumed or computed value, the processors are redesigned for a smaller value of k, which means a smaller number of qubits encoding every node of the graph. This will be illustrated in more detail in chapter 15.

6.3.2. Description of the Solution Tree

6.3.2.1. Basic concepts

The search strategy realizes some design task by seeking to find a set of solutions that fulfill all problem conditions. It checks a large number of partial results and temporary solutions in the tree search process, until finally it determines the optimality of the solutions, the quasi-optimality of the solutions, or just stops when any solution is found.

6.3.2.1. The *state-space* S for a particular problem solved by the program is a set which includes all the solutions to the problem. The elements of S are referred to as *states*. New states are created from previous states by application of operators. During the realization of the search process in the state-space, a memory structure termed *solution tree, solution space*, is used. These states should be not confused with quantum states from the quantum evolution that is executed in the oracle.

6.3.2.1.1. The solution tree is defined as a graph: $D = [NO, RS]$. A solution tree contains *nodes* from set *NO*, and *arrows* from the set of arrows *RS*. Nodes correspond to the stages of the solution process (see Figure 6.3.1.1 and Figure 6.3.1.2.)

6.3.2.1.2. Each arrow is a pair of nodes n_{i1}, n_{i2}. Arrows are also called *oriented edges*. They correspond to transitions from stages to stages of the solution process.

6.3.2.1.3. An *open node* is the node without created *children* , or immediate successors. A child of child is called *grandchild.* If s is a child of p then p is a *parent* of s. A *successor* is defined recursively as a child or a successor of a child. A *predecessor* is defined recursively as a parent or a predecessor of a parent.

6.3.2.1.4. A *semi-open node* is a node that has part of its children created, but not yet all of its children are implicitly formed.

6.3.2.1.5. A *closed node* is a node, where all of its children have already been created in the tree.

6.3.2.1.6. The set of all nodes corresponding to the solutions will be denoted by S_F.

6.3.3. Terminology and Notations

The Sub-Spaces of the Solution Space are related to its structure.

In the solution space we can distinguish the following sub-spaces:

6.3.3.1.***actual solution space*** - the space which has a representation in the computer memory (both RAM and disk),

6.3.3.2.***potential solution space*** - the implicit space that can be created from the actual space using operators and taking into account constraints,

6.3.3.3.***closed space*** - the space which has already been an actual space for some time, but has been removed from the memory (with exception of the solutions).

6.3.3.4.As the search process grows, the actual space is at the expense of the potential space. The closed space grows at the expense of the actual space. The actual space is permanently modified by adding new tree segments and removing other segments. Sometimes the closed space is saved in hard disk, and re-used only if necessary.

6.3.3.5.By "***opening a node***" we will mean creating successors of this node. The way to expand the space, called the *search strategy*, is determined by:

(6.3.5.1) the way the open and semi-open nodes are selected,

(6.3.5.2) the way the operators applied to them are selected,

(6.3.5.3) the way the termination of search procedure is determined,

(6.3.5.4) the conditions for which the new search process is started, and

(6.3.5.5) the way the parts of the space are removed from the memory.

6.3.3.6. The arrows in the tree are labeled by the *descriptors of the operators*. Each node contains a description of a state-space state and some other search-related information. In particular, the state can include the data structure corresponding to the *descriptors of the operators* that can be applied to this node. Descriptors are some simple data items. For instance, the descriptors can be: numbers, names, atoms, symbols, pairs of elements, sets of elements. The choice of what the descriptors are, is often done by the programmer. Descriptors are always manipulated by the search program. (In some problems, they are also created dynamically by the search program.) Descriptors can be stored in nodes or

removed from the descriptions of nodes. As an example of using descriptors, we will discuss the case where the partial solutions are the sets of integers. In this problem then, the descriptors can be the pairs of symbols (*aritmetic_operator, integer*). The application of an operator consists in taking a *number* from the partial solution and creating a new number. This is performed like this:

<new_number> := <number> <aritmetic_operator>< integer>

The *number* is replaced in the partial solution of the successor node by the *new_number*.

6.3.3.7. In those cases that the descriptors are dynamically created, the programs that create them are called the *descriptor generators*. They generate descriptors for each node one-by-one, or all of them at once. The *operators* traverse the tree from a node to a node. Operator is a concept that corresponds to applying certain program to nodes of the solution tree. This program has the descriptor as its parameter. Creating new nodes of the tree is equivalent to searching among the states of *S*.

6.3.3.8. Each of the solution tree's nodes is a *vector of data structures*. For explanation purposes, this vector's *coordinates* will be denoted as follows:

- *N* - the node number,
- *SD* - the node depth,
- *CF* - the node cost,
- *AS* - description of the hereditary structure,

- *QS* - partial solution,
- *GS* - set of descriptors of available operators.

6.3.3.9. Additional coordinates can then be defined, of course, as they are required.

Other notations used:

- *NN* - the node number of the immediately succeeding node (a child),
- *OP* - the descriptor of the operator applied from *N* to *NN*,
- *NAS* - actual length of list *AS*,
- *NQS* - actual length of list *QS*,
- *NGS* - actual length of list *GS*.

6.3.3.10. The operator is denoted by OP_i, and it's corresponding descriptor by r_i. An application of operator OP_i with the descriptor r_i to node *N* of the tree is denoted by $O(r_i, N)$. A *macro-operator* is a sequence of operators that can be applied successively without retaining the temporarily created nodes.

6.4. Formulating a Problem

A prerequisite to formulating the combinatorial problem in the search model is to ascertain the necessary coordinates for the specified problem in the *initial node* (the root of the tree). The way in which the coordinates of the subsequent nodes are created from the preceding nodes must be also found. This leads to the description of the *generator of the solution space (tree generator).* Solution conditions and/or cost functions should be formulated for most of the problems. There are, however, generation problems (such as generating all the cliques of a specific kind), where only the generator of the space is used to generate all the objects of a certain kind.

6.4.1. *QS* is the partial solution: that portion of the solution that is incrementally grown along the branch of the tree until the final solution is arrived at. A set of all possible values of *QS* is a state-space of the problem. According to our thesis, some relation $RE \in S \times S$ of partial order exists usually in *S*. Therefore, the state $s \in S$ symbolically describes the set of all $s' \in S$ such that $s \ RE \ s'$. The solution tree usually starts with $QS(N_0)$ which is either the *minimal* or the *maximal element of S.* All kinds of relations in *S* should be tried to find by the researcher/developer, since they are very useful in creating efficient search strategies.

6.4.2. The set *GS(N)* of descriptors denotes the set of all operators that can be applied to node *N*.

6.4.3. AS(N) denotes the *hereditary structure*. By a hereditary structure we understand any data structure that describes some properties of the node N that it has inherited along the path of successor nodes from the root of the tree.

6.4.4. The *solution* is a state of space that meets all the solution conditions.

6.4.5. The *cost function CF* is a function that assigns the cost to each solution.

6.4.6. The *quality function QF* can be defined as a function of integer or real values pertinent to each node, i.e., to evaluate its quality. It is convenient to define the cost and quality functions such that

$QF(N) \leq CF(N)$ and if *QS(N)* is the solution, then *QF(N) = CF(N)* *Equation 6.4.1*

6.4.7. *TREE(N)* denotes a subtree with node N as the root. Often function *QF(N)* is defined as a sum of function *F(N)* and function $\hat{h}(N)$:

$$QF(N) = CF(N) + \hat{h}(N) \qquad \textit{Equation 6.4.2}$$

6.4.8. $\hat{h}(N)$ evaluates the distance *h(N)* of node N from the best solution in *TREE(N)*. *F(N)*, in such a case, defines a partial cost of *QS(N)*, thus $\hat{h}(N)$ is called a *heuristic function*. We want to define $\hat{h}$ in such a way that it as close to h as possible (see [Nilsson71] for general description and mathematical proofs).

6.4.9. Cost function f

A theoretical concept of function *f* is also useful to investigate strategies as well as cost and quality functions. This function is defined recursively on nodes of the extended tree, starting from the terminal nodes, as follows:

$f(NN) = CF(NN)$ when the terminal node NN is a solution from S_F, *Equation 6.4.3*

$f(NN) = \infty$ when the terminal node NN is not a solution, *Equation 6.4.4*

$f(N) = min\,(f(N_i))$, for all which N_i are the children of node N. *Equation 6.4.5*

This function can be calculated for each node only if all its children have known values, which means practically that the whole tree has been expanded. *f(N)* is the cost of the least expensive solution for the path which leads through node *N*. We assume that the function *CF* can be created for every node *N* (and not only for the nodes from the set S_F, of solutions), it holds that the following must also be true

$$CF(N) \leq f(N) \qquad \textit{Equation 6.4.6}$$

and

$$CF(NN) \geq CF(N) \text{ for } NN \in SUCCESSORS(N) \qquad \textit{Equation 6.4.7}$$

The general idea of the **Branch and Bound Strategy** consists in having a *CF* that satisfies equations 6.4.1, 6.4.2, 6.4.3. Then, knowing a cost CF_{min} of any intermediate solution that is temporarily treated as the minimal solution, one can cutt-off all

subtrees *TREE(N)* for which $CF(N) > CF_{min}$ (or, $CF(NN) \geq CF_{\min}$ when we look for only one minimal solution).

In many problems it is advantageous to use a separate function *QF*, distinct from *CF*, such that *CF* guides the process of cutting-off subtrees, while *QF* guides the selection of nodes for expansion of the tree.

In particular, the following functions are defined:

g(N) the smallest from all the values of cost function calculated on all paths from N_0 to *N*. *Equation 6.4.8*

h(N) the smallest from all the values of increment of cost function calculated from *N* to some $N_k \in S_F$. This is the so-called **heuristic function**. *Equation 6.4.9*

$f(N) = g(N) + h(N)$. *Equation 6.4.10*

Since function *h* cannot be calculated in practice for node *N* during tree's expansion, and *g* is often difficult to find, some approximating functions are usually defined. Function *CF* approximates function *g*. Function $\hat{h}$ approximates function *h*, such that

$$QF(N) = CF(N) + \hat{h}(N) \text{ QF(N)} \qquad \textit{Equation 6.4.11}$$

$$h(N) \geq \hat{h}(N) \geq 0 \qquad \textit{Equation 6.4.12}$$

$$h(M) - h(N) \leq h(M,N) \qquad \textit{Equation 6.4.13}$$

where *h(M,N)* is the smallest of all increment values of cost function from *M* to *N*, when $M, N \notin S_F$. It also holds that:

$$QF(N) = CF(N) \text{ for } N \in S_F \qquad \textit{Equation 6.4.14}$$

$$h(N) \geq \hat{h}(N) = 0 \quad \text{for } N \in S_F \qquad \textit{Equation 6.4.15}$$

Functions defined like this are useful in some search strategies, called *Nilsson A* Search Strategies*. Sometimes while using branch-and-bound strategies it is not possible to entirely define the cost function *g(N)* for $N \notin S_F$. However, in some cases one can define a function *QF* such that for each *N*.

$$QF(N) \leq g(N) \qquad \textit{Equation 6.4.16}$$

For nodes $N \in S_F$ one calculates then *g(N) = CF(N)*, and then uses standard cut-off principles, defining for the remaining nodes N_i: $CF(N_i) = QF(N_i)$, and using function *CF* in a standard way for cutting-off. A second, standard role of *QF* is to control the selection of non-closed nodes. (By non-closed nodes we mean those that are either open or semi-open.) One should then try to create QF that plays both of these roles.

A *quasi-optimal* or *approximate solution* is one with no redundancy; i.e., if the solution is a set, all of its elements are needed. When the solution is a path in a certain graph, for example, it has no loops. An *optimal solution* is a solution $QS(N) = s \in S$ such that there does not exist $s' \in S$ where *QF(s) > QF(s')*. The problem can have

more than one optimal solution. The set of all solutions will be denoted by *SS*. Additional *quality functions for operators* can also be used.

6.4.10. Descriptors and tree types

In many combinatorial problems, the set of all mathematical objects of some type are needed: sets, functions, relations, vectors, etc. For example, the following data are created:

- The set of all subsets of prime implicants in the minimization of a Boolean function.
- The set of all subsets of bound variables in the Variable Partitioning Problem.
- The set of all two-block partitions in the Encoding Problem.
- The set of maximal compatibles in the Column Minimization Problem.

These sets can be generated by the respective search routines created for them in a standard way, that use the generators of trees. This is useful in *direct problem descriptions*.

It is desirable to develop descriptor generators for several standard sets, several types of tree generators, many ordering possibilities for each generation procedure, and several tree extension strategies for each ordering. The *type of tree* is defined by two generators: the generator that creates the descriptors, and the generator that generates

the tree. A *full tree* is a tree created by the generators only, ignoring *constraint conditions*, quality functions, dominations, etc. Full trees of the following types exist:

- T_1 - a tree of all subsets of a given set,
- T_2 - a tree of all permutations of a given set,
- T_3 - a tree of all one-to-one functions from a set A to set B.

and many others.

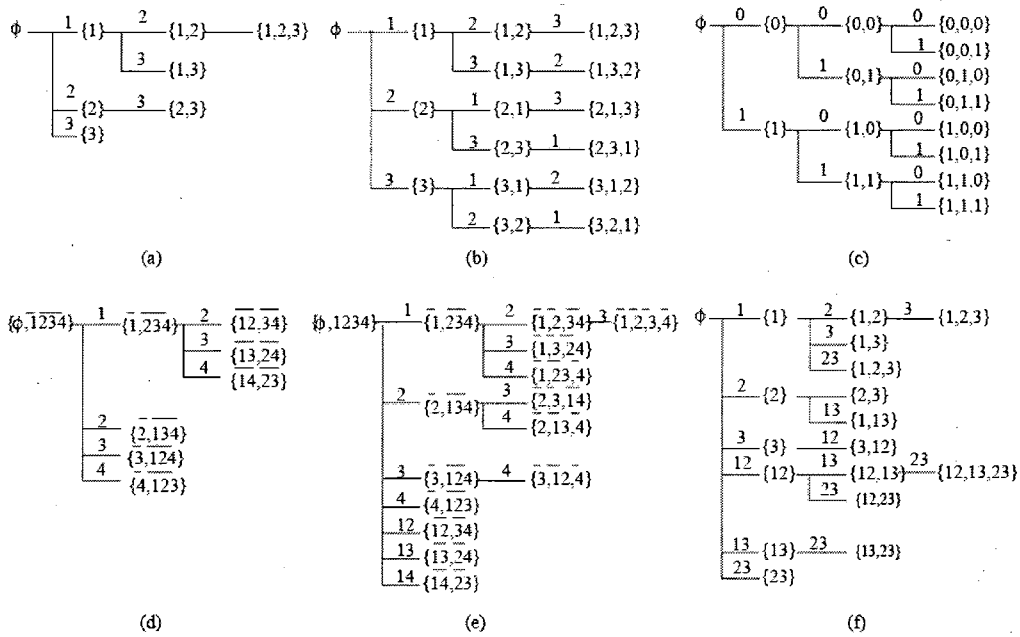


Figure 6.4.10.1: Examples of tree generators.

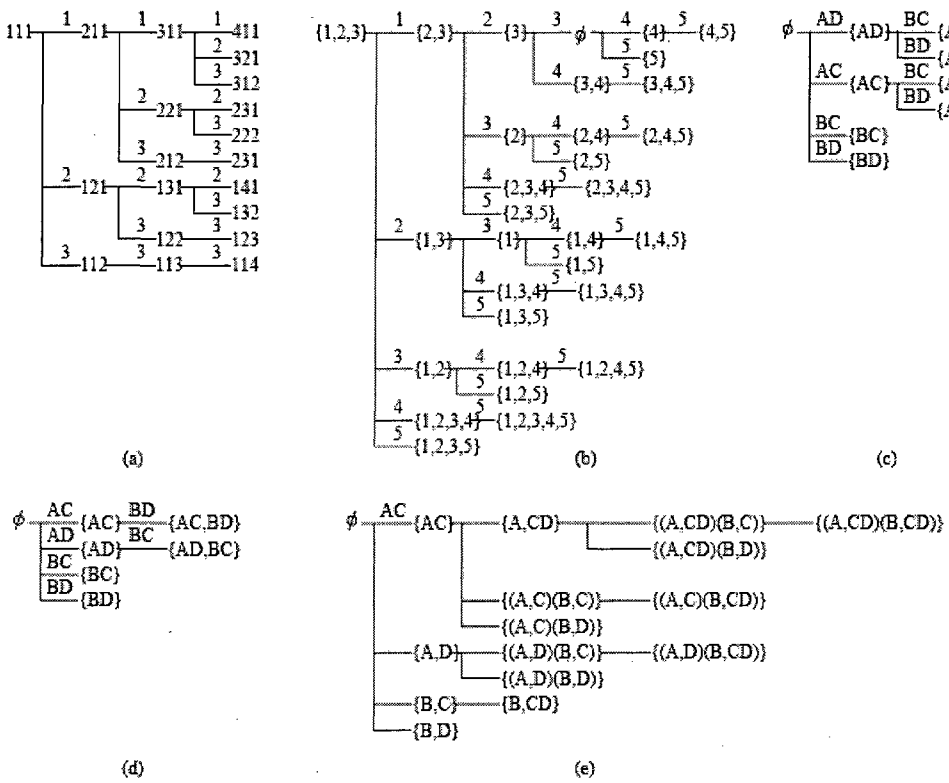


Figure 6.4.10.2: More examples of tree generators.

The type T_1 tree generator of the full tree of the set of all set's {1, 2, 3} subsets, as shown in Figure 6.3.1, can be described as follows.

1. *Initial node*} (root) is described as:

$$QS(N_0) = \phi; \qquad \textit{Equation 6.4.10.1}$$

$$GS(N_0) = \{1,2,3\}; \qquad \textit{Equation 6.4.10.2}$$

where ϕ is an empty set, and N_0 is the root of the tree.

2. In a recursive way, the *children* of any node N are described as follows:

$$(\forall r \in GS(N))[QS(NN) = QS(N) \cup \{r\}; GS(NN) = \{r_1 \in GS(N) \mid r_1 > r\}]$$

Equation 6.4.10.3

where *NN* is some child of node *N*, and *r* is the descriptor of the operator that creates the new nodes in this tree. Set *GS* is either stored in the node or its elements are generated one by one in accordance with the ordering relation > while calculating the children nodes.

Figure 6.4.10.1 and Figure 6.4.10.2 present examples of full trees for many important combinatorial problems. They show the partial solutions in nodes and the descriptors near arrows.

The trees in Figure 6.4.10.1 are:

(a) the tree of all subsets of set,

(b) the tree of all permutations of a set,

(c) the tree of all binary vectors of length 3,

(d) the tree of all two-block partitions of set {1, 2, 3, 4},

(e) the tree of all partitions of set {1, 2, 3, 4},

(f) the tree of all covers of set {1, 2, 3, 4} with its subsets.

The trees in Figure 6.4.10.1 and 6.4.10.2 are the following. Figure 6.4.10.1 a presents the tree for all 3-element numerical vectors, such that they sum is a constant in every level. In the first level the sum is 3, in the second level the sum is 4, in the third level

the sum is 5, in the fourth level the sum is 6. Figure 6.4.10.2 presents the tree of all subsets of set {1, 2, 3, 4, 5}.The tree generates levels of equal distance from the subset {1, 2, 3}, in the second level there are subsets that differ by one from {1, 2, 3}. All descriptors from set {1, 2, 3, 4, 5} are checked in the first level. If the descriptor is in the subset, it is subtracted, if the descriptor is not in the subset, it is added. In all next levels, the sets of descriptors for each node are created in exactly the same way as in the standard tree for all subsets, that have been shown in detail in Figure 6.4.2. Others can be explained in a similar way.

6.4.11. Encoding for GA and Search Algorithms to synthesize quantum circuits.

Genetic algorithm is used as a component in our general search framework. We do not explain it as it is popularly known.

As an illustration, in this section we introduce a notation that will be useful to explain not only genetic algorithm but also search and other algorithms to synthesize quantum circuits in a systematic and uniform way. Let us denote the whole column of a 3×3 quantum array by a symbol. For instance, symbols A, B and C are used in Figure 6.4.11.1 below to denote the Feynman gate with EXOR down, the Feynman gate with EXOR up and the Toffoli gate with lowest bit as the target (the bit with exoring), respectively.

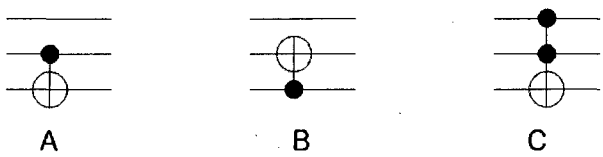


Figure 6.4.11.1: Symbols for columns of a Quantum array used to encode genes in a chromosome of a GA for 3×3 quantum arrays synthesis. For instance the Toffoli gate controlled from two top qubits is denoted by capital letter C.

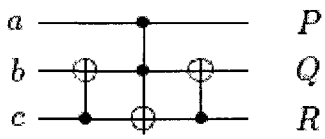


Figure 6.4.11.2: Circuit corresponding to the Chromosome BCB, which is the quantum circuit for the Fredkin gate composed from two Feynman gates and the Toffoli gate.

The functional circuit from Figure 6.4.11.2 (the phenotype) corresponds to the Chromosome BCB (genotype).

Figure 6.4.11.3 illustrates hypothetical operation of the Genetic Algorithm to find the circuit from Figure 6.4.11.2. The analysis/simulation method as in Chapter 2 is used to calculate the fitness function and to verify the correctness of the solution genotype circuit from Figure 6.4.11.2. Figure 6.4.11.4 illustrates the operation of the exhaustive breadth-first search algorithm for the same task. As we see, the GA and the tree are just two different strategies to search the same space of character strings. Our intelligent learning algorithm from this chapter 6 uses these two "pure" search methods, many other methods and also combined search methods as its special cases.

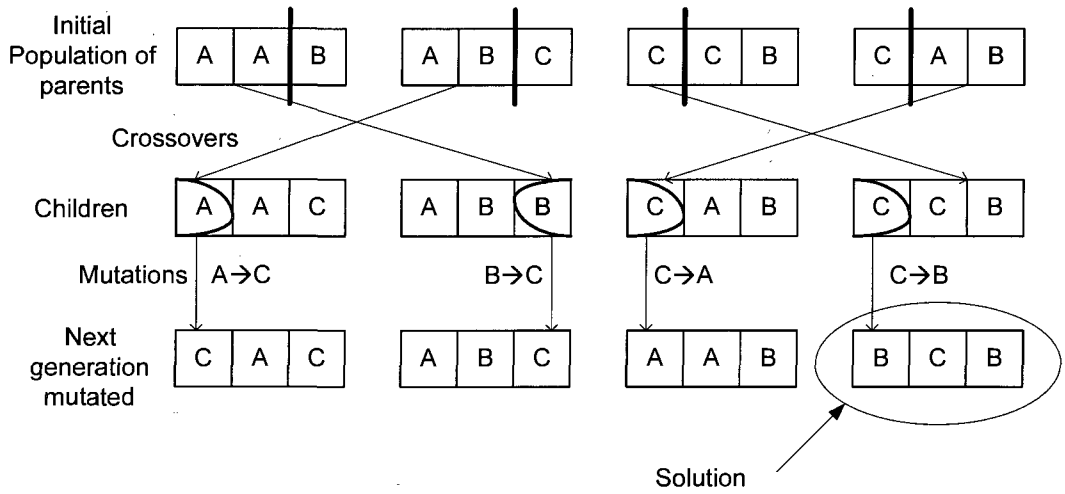


Figure 6.4.11.3: Operation of the Genetic Algorithm to find the chromosome BCB leading to the phenotype circuit from Figure 6.4.11.2.

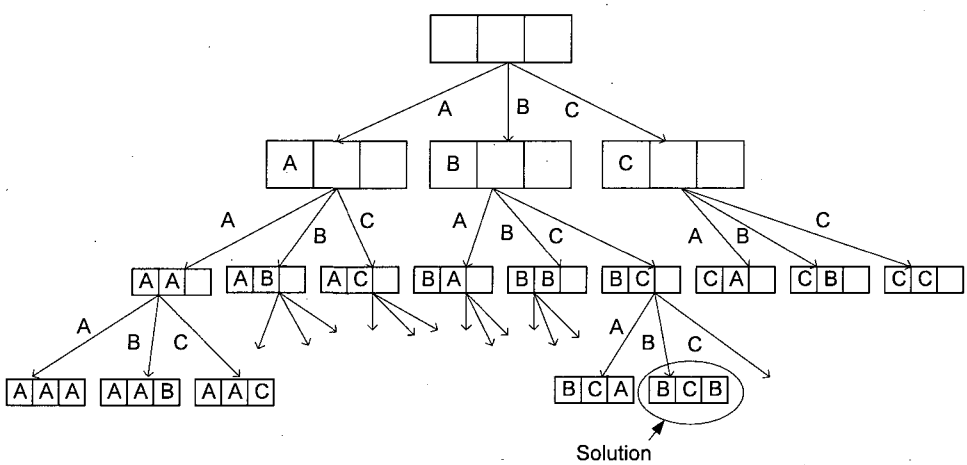


Figure 6.4.11.4: Operation of the exhaustive breadth first search algorithm to find the circuit from Figure 6.4.11.2. The fitness function uses as its component the circuit's cost function which is the same as in the GA.

6.5. Creating Search Strategies

A number of *search strategies* can be specified for the tree search procedure along with the quality functions. Beginning with the initial node, the information needed to produce the solution tree can be divided into *global information*, that relates to the whole tree, and *local information*, that is concerned only with local subtrees. Local information in node *N* refers to subtree *TREE(N)*. The developer-specified search strategies are, therefore, also divided into a *global search* and a *local search*. The selection of the strategy by the user of the Universal Search Strategy from section 6.6 is based on a set of *strategy describing parameters*. By selecting certain values, the user can, for instance, affect the size of subsequent sets of bound variables or the types of codes in the encoding problem. We assume also that in the future we will create smart strategies that will allow to dynamically change the strategy parameters by the main program during the search process. Such strategies, that for instance search breadth-first and after finding a node with certain properties switch to depth-first search, have been used with successes in Artificial Intelligence. Let us distinguish the *complete search strategies* that guarantee finding all of the optimal solutions from the *incomplete strategies* that do not. Both the complete and the incomplete search strategies can be created for a complete tree search method. A tree searching strategy that is created for a complete tree search method and includes certain restricting conditions or cutting-off methods that can cause the loss of all optimal solutions is referred to as an *incomplete search strategy* for a complete search method. By removing such conditions a complete search strategy is restored, but it is less efficient.

This approach offers the following advantages:

6.5.1. The *quasi-optimal solution* is quickly found and then, by backtracking, the successive, better solutions are found until the *optimal solution* is produced. This procedure allows to investigate experimentally the trade-offs between the quality of the solution and speed of arriving at it.

6.5.2. The search in the state-space can be limited by including as many *heuristics* as required. In general, a heuristic is any rule that directs the search. It will be more on the heuristics in the sequel.

6.5.3. The application of various quality functions, cost functions, and constraints is possible.

6.5.4. The problem can be described within several degrees of accuracy. The *direct description* is easy for the designer to formulate, even though it produces less efficient programs. It is created in the early prototype development stages, on the basis of the problem formulation only, and the heuristics are not yet taken into account. The only requirement is that the designer knows how to formulate the problem as a state-space problem using standard mathematical objects and relations. Only the standard node coordinates are used. The *detailed description of the tree search method*, on the other hand, provides the best program that is adequate for the specific problem but it requires a better understanding of the problem itself, knowledge about the program structure, and experimentation.

6.5.5. By using macro-operators along with other properties, the main strategies require less memory than the comparable, well-known search strategies [Nilsson71, Perkowski76].

6.6. General Strategies for search.

The search strategy is either selected from the *general strategies*, of which the following is a selection, or it is created by the developer's writing of the *sections codes*, and next the user assigning values to the *strategy describing parameters*.

6.6.1. Breadth-First. With this strategy, each newly created node is appended to the end of the so called *open-list* which contains all the nodes remaining to be extended: *open nodes*. Each time the first node in the list is selected to be extended, it is removed from the list. After all the available operators for this node have been applied, the next node in the open-list to be extended is focused on.

6.6.2. Depth-First. The most recently generated node is extended first by this strategy. When the specified depth limit SD_{max} has been reached or some other cut-off condition has been satisfied, the program backtracks to extend the deepest node from the open-list. This newly created node is then placed at the

beginning of the open-list. The consequence is that the first node is also always the deepest.

6.6.3. Branch-and-Bound. The temporary cost *B* is assigned which retains the lowest cost of the solution node already found. Whenever a new node *NN* is generated, its cost *CF(NN)* is compared to the value of *B*. All the nodes whose costs exceed the value of *B* will be cut off from the tree.

6.6.4. Ordering. This strategy, as well as the next one, can be combined with the Branch-and-Bound strategy. A quality function *Q(r, N)* is defined for this strategy to evaluate the cost of all the available descriptors of the node being extended. These descriptors are applied in the operators in an order according to their evaluated cost.

6.6.5. Random. With this strategy, the operator or the open node can be selected randomly for expansion, according to the probability distribution specified.

6.6.6. Simulated annealing. This strategy transforms nodes from open list using the respective algorithm. This strategy is known from literature and will be no further discussed here.

6.6.7. Genetic algorithm. This strategy uses open list as a genetic pool of parents' chromosomes.

6.6.8. Quantum Grover Search. This is exhaustive strategy presented in Chapter 5. It can be simulated in standard software by exhaustive search based on standard combinational oracle.

(In case of ECPS which is the special case of QSPS all the strategy creating tools should be defined as C++ classes.)

The strategy describing subroutines and parameters are outlined in section 6.7 below.

6.7. Conditions in QSPS.

There are two types of conditions for each node of the tree: *by-pass condition* and *cut-off condition*. The cut-off condition is a predicate function defined on node N as an argument. If the cut-off condition is met in node N, the subtree *TREE(N)* is prevented from being generated and backtracking results. The by-pass conditions do not cause backtracking and the tree will continue to extend from node N. The following cut-off conditions exist:

6.7.1. Bound Condition. This condition is satisfied when it is found (possibly from information created in node N) that there exists node N_1 (perhaps not yet constructed) such that $CF(N_1) < CF(N)$ and $QS(N_1)$ is a solution.

6.7.2. Depth Limit Condition. This condition is satisfied when *SD(N)* is equal to the declared depth limit SD_{max}.

6.7.3. Dead Position Condition. This condition is satisfied when no operators can be applied to N, i.e. $GS(N) = \phi$.

6.7.4. Restricting Conditions. Each of these conditions is satisfied when it is proved that *QS(N)* does not fulfill certain restrictions, i.e. no solution can be found in *TREE(N)*, for one or another reason.

6.7.5. Solution Conditions of the Cut-Off Type. Any of these conditions is satisfied when the property of the problem is that if *QS(N)* is a solution, then for each $M \in TREE(N)$, $CF(M) > CF(N)$ (or $CF(M) \geq CF(N)$). Therefore, node *M* may be not taken into account.

6.7.6. Branch Switch Conditions and **Tree Switch Condition.** Satisfaction of Switch Condition causes modification of the actual search strategy to another strategy, resulting from the search context and previous conditional declaration of the user. This leads to the so-called *Switch Strategies* that dynamically change the search strategy during the process of search. For instance, the depth first strategy can be changed to breadth-first if certain condition is met.

6.7.7. Other types of conditions. They are formulated for some other type restrictions special to problems (selected by the user by setting flags in the main algorithm).

A value that interrupts the search when a solution node *N* is reached such that $CF(N) = CF_{min\ min}$ is denoted by $CF_{min\ min}$. This is a known minimum cost of the solution. This value can be arrived in many ways, usually it comes from a calculated guess, or is derived by some calculation or by mathematical deduction. It may also be a known optimal cost. In most cases the value is a "guessed value", that may be incorrect. Therefore, it will serve here only as one more control parameter.

When all the solution conditions are met in a certain node *N*, *QS(N)* is a solution to the given problem. This is then added to the set of solutions and is eventually printed. The value of *CF(N)* is retained. If one of the solutions is a "cut-off type solution", then the program backtracks. Otherwise, the branch is extended.

Similar strategies are used in case of parallel quantum programs/oracles. The only difference is that in quantum the granularity of search is with accuracy to whole subtrees and not to single nodes with their successors. In theory, the granularity in quantum search can be also to small trees of a nodes with all its successors. For instance, the quantum computer may find all Boolean functions created from some function Fi by exoring it with all possible products of literals of some type. This will be illustrated in examples.

6.8. Relations on Operators and States

Determining some relations on operators (descriptors) is often very useful. Similarly, the developer may determine certain relations on states of the solution space, or on the nodes of the tree.

Having such relations allows to cut-off nodes. It allows also to remove dispensable descriptors from the nodes. Specifically, in many problems it is good to check solution conditions immediately after creating a node, and next immediately reduce the set of descriptors that can be applied to this node.

The following relations between the operator descriptors (so called *relations on descriptors*) can be created by the program developer to limit the search process:

- *relation of domination,*
- *relation of global equivalence,*
- *relation of local equivalence.*

We will define *local* and *global* domination relations. Operator O_1 is locally dominated in node N by operator O_2 (or descriptor r_2 is locally dominated by r_1) when:

$$O_1, O_2 \in DOML(N) \qquad \textit{Equation 6.8.1}$$

while relation *DOML* satisfies the following conditions:

$$DOML \text{ is transitive} \qquad \textit{Equation 6.8.2}$$

and

$$(O_1, O_2) \in DOML \Rightarrow f(O_1(N) \leq f(O_2)) \qquad \textit{Equation 6.8.3}$$

We will apply the notation:

$$(O_1, O_2) \in DOML \overset{define}{\Leftrightarrow} O_1 \overset{\succ}{\underline{L}} O_2 \qquad \textit{Equation 6.8.4}$$

We will define operator O_2 as *locally subordinated* in node N with respect to operator O_1 (where $r_1, r_2 \in GS(N)$), if

$$O_1 \overset{\succ}{\underline{L}} O_2 \wedge O_2 \overset{\prec}{\underline{L}} O_1 \qquad \textit{Equation 6.8.5}$$

This will be denoted by

$$O_1 \overset{\succ}{L} O_2 \qquad \textit{Equation 6.8.6}$$

If $O_1 \overset{\succ}{L} O_2$ in node N, then tree TREE($O_2(N)$) can be cut-off without sacrificing optimal solutions, since

$$f(O_1(N)) < f(O_2(N)) \qquad \textit{Equation 6.8.7}$$

It is easy to check that relation $\overset{\approx}{L}$, defined as

$$O_1 \overset{\approx}{L} O_2 \quad \overset{define}{\Leftrightarrow} \quad O_1 \overset{\succ}{\underline{L}} O_2 \wedge O_2 \overset{\succ}{\underline{L}} O_1 \qquad \textit{Equation 6.8.8}$$

is an *equivalence relation*, which we will call the *Local Relation of Equivalence of Descriptors* in node *N*. Relation $\overset{\approx}{L}$ partitions set *GS(N)* into classes of abstraction $[r_i]$. It is obvious from these definitions, that when one wants to obtain only a single optimal solution being a successor of *N*, then from each class of abstraction $[r_i]$ only one element should be selected. All remaining elements should be removed from *GS(N)*.

The relation of *global domination* gives better advantages than the local domination, in cases that such a relation of global domination can be defined. Operator O_2 is *globally dominated* in tree *TREE(N)* by operator O_1 when

$$(O_1, O_2) \in DOML(N) \subset O(TREE(N)) \times O(TREE(N)) \qquad \textit{Equation 6.8.9}$$

By *O(TREE(N))* we denote the set of operators to be applied in tree *TREE(N)*. Relation *DOMG* satisfies the following conditions:

$$DOMG \text{ is transitive} \qquad \textit{Equation 6.8.10}$$

and

$$\begin{aligned}&(O_1, O_2) \in DOMG(N) \Rightarrow (\forall M_1 \in NO(TREE(N)))\\ &[r_1 \in GS(M_1) \wedge r_2 \in GS(M_1) \wedge f(O_1(M_1) \in f(O_2(M_1)) \vee r_2 \notin GS(M_1)]\end{aligned} \qquad \textit{Equation 6.8.11}$$

Similar to local relations, one can define relation $\tilde{\tilde{G}}$ of *global subordination in tree TREE(N)*, and relation $\tilde{\tilde{G}}$ of *global equivalence* in tree *TREE(N)*.

Relation $\tilde{\tilde{G}}$ partitions every set *GS (M_I)* for each $M_I \in NO(TREE(N))$ into classes of abstraction.

If we have no intention to find all optimal solutions, then from each class of abstraction we take just one element, and the remaining operators are removed from *GS (M_I)*.

The following theorem can be proven.

Theorem 6.8.1. Let us denote by $[r_i]$ the global equivalence class of operator O_I in node *N*. If for each branch *N*, N_1, N_2 ,.... N_k of tree *TREE(N)* it holds $GS(N) \supseteq GS(N_2) \supseteq GS(N_k)$ then the descriptors from set $[r_i] \setminus r_i$ can be immediately removed from all sets GS in all nodes in *TREE(N)*.

When we want to use the relation of global equivalency in certain node N, and the property from this theorem does not hold, then it is necessary to calculate the descriptors, which should be not applied in node N (sometimes it can be easily done from an analogous set for the node being the parent of this node).

Node M is *dominated* by node N if $f(N) \leq f(M)$

$$(N,M) \in DOMS \Leftrightarrow f(N) \leq f(M) \qquad \textit{Equation 6.8.12}$$

Similarly as before, we can introduce relations $\overset{\succ}{-}, \overset{\succ}{s}$ and $\overset{\approx}{s}$.

If ST_1 and ST_2 are two strategies, which differ only in their domination relations D_1 and D_2 (these can be relations of domination of any of the presented types) and if $D_1 \supset D_2$ then $k_i^1 \leq k_i^2$ for each of the introduced coefficients k_i.

Observe, that by incorporating the test for the relation of domination (or equivalence) to an arbitrary strategy that generates all optimal solutions, there exists the possibility of sacrificing only some optimal solutions (or all the optimal solutions but one). This decreases the number of generated nodes, which for many strategies is good both with respect to the reduced time, and reduced memory. On the other hand, if evaluating relations is very complex, the time of getting the solution can increase. The stronger is the domination relation, the more complicated is its evaluation, or the larger is its domain. Therefore, the time for testing domination would grow. In turn, the more gain from the decreased number of generated nodes. Often it is convenient to investigate

relation of domination only in nodes of the same depth, or on operators of some group. Theoretical analysis is often difficult and experimenting is necessary.

Finally, observe that domination relations are not based on function f, because the values of f are not known a'priori, while creating the levels of the tree. The domination relations are also not based on costs, but on some additional problem-dependent information of the program, about the nodes of the tree. These relations come from certain specific problem-related information. In most cases, the implication symbol in Equation 6.8.3 cannot be replaced by the equivalence symbol, since this would lead to optimal strategies with no search at all, and each branch would lead to optimal solutions.

6.9. Component Search Procedures of C++ realization of ECPS.

The **Main Universal Search** subroutine of a search program is in charge of the global search. It takes care of the selection of strategies, the arrangement of the open-list and the other lists as well as the decision making facilities related to the cut-off branch, and the configuration of the memory structures to store the tree. The lines of code that realize the strategies of breadth-first, depth-first, or branch-and-bound are built into the main search routine. Subroutines *RANDOM1* and *RANDOM2* are selectively linked for the random selection of the operator or the open node, respectively. The role of the subroutines linked to the Universal Search subroutine is as follows:

- *GENER* is responsible for the local search that extends each node. *GENER* cuts off the nodes which will not lead to the solution node when the description for the new node is created.
- *GEN* carries out the task of creating nodes.

Other subroutines, offered to create local search strategies, are the following:

- *MUSTAND* and *MUSTOR* are subroutines that serve to find two types of the so-called *indispensable operators*. (The indispensable operators are the operators that must be applied). All operators found are indispensable in the *MUSTAND* subroutine, and only one of operators is indispensable in case of the *MUSTOR* subroutine. The set of indispensable operators is next substituted as the new value of coordinate *GS(N)*.
- subroutine *MUSTNT* deletes *subordinate operators*. Subordinate operators are those that would lead to solutions of higher costs, or to no solutions at all. The set *MUSTNT(N)* is subtracted from set *GS(N)*.

Domination and equivalence conditions for the tree nodes can also be declared as follows:

- *EQUIV* cancels those nodes that are included in other nodes.
- *FILTER* checks whether the newly created node meets the conditions.
- *SOLNOD* checks the solution condition.

- *REAPNT* is used to avoid the repeated applications of operators when the sequence of operator applications does not influence the solution.

These local strategies, as well as the global strategies listed above, can be selected by reading the parameter values as input data. *ORDER* sorts the descriptors, *QF* calculates the quality function for the descriptors, and *CF* calculates the cost of the nodes.

6.9.1. Universal Search Strategy

In this section we will present the universal search strategy. First we will explain the meaning of all variables and parameters. Next the pseudo-code of the main strategy subroutine will be given, followed by the pseudo-code of its GENER subroutine.

6.9.1.1. Meaning of Variables and Parameters

CF_{min} - cost of the solution that is actually considered to be the minimal one. After a full search, this is the cost of the exact minimum solution.

SOL - set of solutions actually considered to be minimal. If parameter *METHOD* = 1, then this set has always one element. When a full search has been terminated, this set includes solutions of the exact minimal cost.

OPERT - list of descriptors, which should be applied to the actual state of the tree.

OPEN - list of open and semi-open nodes.

N - actual state of the space.

NN - next state of the space (this state is actually being constructed from node *N*).

OUTPUT - a parameter that specifies the type of the currently created node;

- when *OUTPUT* = 0, the created node *NN* is a branching node;
- when *OUTPUT* = 1, the created node *NN* is an end of a branch;
- when *OUTPUT* = 2, a *quasioptimal solution* was found, whereby by a quasioptimal solution we understand any solution that has the value of the cost function not greater than the user-declared parameter $CF_{min\ min}$.

$CF_{min\ min}$ - a parameter assumed by the user, determined heuristically or methodically, the value that satisfies him.

QF_{min} - the actually minimal value of the quality function.

OPT - a parameter. When OPT = 1, then any solution is sought, otherwise the minimal solution.

PP9 - a parameter. When *PP9* = 1, then the subroutine *"Actions on the Selected Node"* is called.

EL - actual descriptor from which the process of macro-generation starts (this is the first element of list *OPERT*).

DESCRIPTOR - actual descriptor during the macrogeneration process.

MUST - list of descriptors of operators, which must be applied as part of the macrooperator.

PG5 - a parameter. If *PG5* = 1, then it should be investigated, immediately aftear the creation of node *NN*, if there exists a possibility of cutting-off node *NN*.

PG6 - a parameter. If *PG6* = 0, then it should be investigated if node *NN* can be cut-off with respect to the monotonically increasing cost function *CF*, and in respect to satisfaction of $CF_{min} = CF(NN)$.

PG6D - a parameter. If *PG6D* = 1, then value CF_{min} should be calculated with respect to a subroutine of a user, otherwise CF_{min} is calculated in a standard way as *CF(NN)*.

PG6E - a parameter. If *PG6E* = 1, then the learning subroutine is called.

PG6F - a parameter. If *PG6F* = 1, then after finding a solution the actions declared by the user are executed.

PG7 - parameter; if *PG7* = 1 then descriptors defined by other parameters are removed from *GS(NN)*.

6.9.1.2. The Main Search Strategy

1. Set the parameter variables to the values that will determine the search strategy.
2. $CF_{min} := \infty$, $SOL := \emptyset$, $OPERT := \emptyset$, $OPEN := \emptyset$.

3. Call the macrogeneration subroutine *GENER* for the user-declared initial state N_0.

4. If the value of variable *OUTPUT* (this value is set by subroutine *GENER*) is 1 or 2 then, according to the declared parameters, return to the calling program for the problem, or select a strategy corresponding to the declared data.

5. State N_0 has been (possibly) transformed by subroutine *GENER*. Store the new state in the tree. $OPEN := N'_0$.

6. If $OPEN = \emptyset$ then either return to the calling program, or change the search strategy, according to the parameters and the strategy change parameters for trees (*Tree~Switch*). (see section 6.2.8).

7. If the threshold values for the tree have been exceeded (size, time, etc) then return to the calling program, or change strategy, as in step 6. If the *Stopping Moment Learning Program* decides termination of the search, then this search process is terminated. Return to the calling program, that will decide what to do next (see section 2).

8. $N :=$ selected node from list *OPEN*. This step is executed on the basis of Strategy Selecting Parameters, including minimal values *QF* or *CF*. If parameters specify A^* Strategy of Nillsson and $QF(N) \geq QF_{min}$, then return to the calling program (since all minimal solutions have been already found).

9. If parameter $PP9 = 1$, then call subroutine "Actions on the selected node" (this subroutine can, for instance, declare such actions as: (1) cutting-off a node

upon satisfying some condition, (2) sorting *GS(N)*, (3) assigning *GS(N)* := Ø,(4) deleting redundant or dominated operators). *OPERT* := *GS(N)*. Remove from list OPEN all closed nodes.

10. If *OPERT* = Ø then go to 6.
11. *EL* := *OPERT*[0], remove *EL* from list *OPERT*. (*OPERT*[0] selects the first element of list *OPERT*)
12. Call subroutine *GENER*.
13. If a *Branch Switch Strategy* has been declared and a respective switch condition is satisfied then execute the Branch Switch type modification of the search strategy.
14. If *OUTPUT* = 0, then store the node *NN* (created earlier by subroutine *GENER*) in the tree (if a tree data structure is used in addition to list *OPEN*). Insert this node in certain position in list *OPEN*. This position depends on the selected strategy. If *OUTPUT* = 2, then (if parameter *OPT* = 2 then return to the calling program, else go to 11).
15. Go to 11.

6.9.1.3. Subroutine GENER

1. If *GENER* is executed in step 13 of the main search strategy then MUST: = EL (value of *EL* has been previously set in the main search routine).

2. If *MUST* = Ø, then set *OUTPUT*: = 0, return.

DESCRIPTOR := *MUST*[0].

3. Call subroutine *OPERATOR* written by User. We denote this by *O(N, DESCRIPTOR)*. This call generates the new state *NN*, for the *DESCRIPTOR* selected in step 3. *GS(N)* := *GS(N)* \ *DESCRIPTOR* (i.e. DESCRIPTOR is removed from GS(N)).

4. If parameter *PG5* = 1

   and

   (node *NN* satisfies on of the Branch Cut-Off Conditions **or** *NN* is dominated by another node), then cut-off node *NN*. *OUTPUT* := 1. Return.

   (the above condition means that node *NN* is equal to another node, or node *NN* is dominated by another node based on one of the relations: Node Domination, Node Equivalence, Node Subordination).

   If *CF(NN)* > CF_{min} (while looking for all minimal solutions)

   or

   If *CF(NN)* ≥ CF_{min} (while looking for a single minimal solution),

   then

   cut-off node *NN*. *OUTPUT* := 1, return.

5. If parameter *PG6* = 0 then

   If CF_{min} = *CF(NN)* and the parameter specifies that *CF* is monotonically increasing and node *NN* does not satisfy all the user-declared Solution Conditions, then cut-off node *NN*, *OUTPUT* := 1,

return. If node *NN* satisfies all the user-declared Solution Conditions, then

A. If $CF(NN) \leq CF_{min}$ and the A^* Strategy of Nillsson is realized, then store $QF_{min} := QF(NN)$.

B. If $CF(NN) = CF_{min}$, then

   i. if all optimal solutions are sought,
   then append *QS(NN)* to the list of solutions *SOL* else do nothing.

C. If $CF(NN) < CF_{min}$ then set *SOL* := QS(NN).

D. If parameter *PG6D* = 1, then calculate CF_{min} using the User Subroutine Calculating CF_{min}, else $CF_{min} := CF(NN)$.

E. If parameter *PG6E* = 1, then call the subroutine "Parametric Learning the Quality Function for Operators".

F. If parameter *PG6F* = 1, then call the subroutine "Actions after Finding a Solution". This is a subroutine used to specify the actions to be executed after the solutionis found. These actions can be: printout, display, storage, etc.)

G. If $CF(NN) = CF_{min}$, then *OUTPUT* := 2, return.

H. If $CF(NN) \neq CF_{min\,min}$, then *OUTPUT* := 1, return.

6. If *PG7* = 1, then remove the indispensable descriptors from *GS (NN)*. Depending on the values of parameters, the following types of descriptors are being removed:

   (2) Inconsistent Descriptors,

   (3) Descriptors that result from:

   (3a) Local Subordination Relation,

   (3b) Local Domination Relation,

   (3c) Local Equivalence Relation,

   (3d) Local Equivalence Relation,

   (3e) Global Subordination Relation,

   (3f) Global Domination Relation,

   (3g) Global Equivalence Relation.

   Use subroutine *MUSTNT*.

   If a Condition of Node Expansion Termination is satisfied then set *GS(NN)* := Ø. If the set of Indispensable Operators of *MUSTOR* type is declared and

   respective Condition of operators of *MUSTOR* type is satisfied, then set *GS(NN)* := *MUSTOR(GS(NN))*.

1. *N* := *NN*.

2. If *MUST* $\neq$ Ø, then go to 2.

   else *MUST* := set of Indispensable Descriptors of *MUSTAND* type in *GS(N)*.

   Go to 2.

The first call of subroutine *GENER* is intended to check if the indispensable operators of type *MUSTAND* exist in the initial state given by the user. These operators are applied to the successively created states, until a solution is found, or a node is found, in which no longer exist any indispensable descriptors. When subroutine *GENER* is returned from, the state N_0 may have been transformed. The condition to find the minimal solution is to terminate with empty list *OPEN*. In steps 8 and 9, with respect to the strategy determining parameters, the node for expansion is selected, together with the operators that will be applied to this node. This node can be the open or semi-open type. *Open* means all possible operators have been applied to it. *Semi-open*, means some operators (descriptors) were applied but other descriptors remain, ready to be applied in a future. Selected descriptors are successively applied to the node, until list *OPERT* is cleared.

The value of parameter *OPT* is determined by the user. If *OPT* = 1, then the subroutine will return to the calling program after finding the first quasi-optimal solution.

Subroutine *GENER* is used to find and apply macro-operators. Descriptor *EL*, selected in the main search program, is put to list *MUST* of indispensable descriptors (except of the call in step 3).Such approach has been chosen in order to check if some indispensable descriptors exist in the initial state. It is known for all subsequent nodes

that there are no indispensable descriptors, since if there were an indispensable descriptor in a node created by *GENER*, it would be immediately applied. Therefore, the result of subroutine *GENER* is always a single child, that has no indispensable operators.

In a general case, pure branch-and-bound strategy (discussed below) will terminate in steps 4 and 6 of the main search strategy. The *A** strategy of Nilsson will terminate in step 8.

Of course, in lists OPEN, OPERT and other lists, not objects are stored, but pointers to them.

6.10. Pure Search Strategies

In this section we present the so-called pure search strategies. They will not require strategy-switching. Many of these strategies are known from the literature. Pure strategies are the following.

1. Strategy ST_{QF} is defined as follows:

$$QF(SEL1_{QF}(OPEN)) = min_{Ni \in OPEN}\ QF(N_i), \quad SEL2(x) = x$$

Equation 6.10.1

SEL1 is the node selection strategy and *SEL2* is the descriptor selection strategy. In this strategy, all children of node *N* are generated at once. This corresponds to the "Ordered Search" strategy, as described in [Nilsson71, Ibaraki76].

If, in addition to the above formula 6.10.1 function *QF* satisfies conditions 6.4.8, 6.4.9, 6.4.10, 6.4.11, 6.4.12 then it corresponds to the well-known *A** strategy of Nilsson.

2. Strategy ST_{CF} (strategy of equal costs), in which:

   $CF(SEL1_{CF}(OPEN)) = min_{Ni \in OPEN}\ CF(N_i), \quad SEL2(x) = x$

   Equation 6.10.2

3. This is a special case of the strategy from point 1.

4. Depth-first Strategy $SEL1_d(OPEN)$ = the node that was recently opened, $SEL2(x) = x$

5. Breadth-first Strategy

   $SEL1_b(OPEN)$ = the first of the opened nodes, $SEL2(x) = x$

6. Strategy $ST_{d,s,k}$ (depth, with sorting and selection of k best operators) $SEL1_{d,s,k}(NON\text{-}CLOSED)$ = the node that was recently opened, $SEL2_{d,s,k}(GS(SEL1(NON\ -\ CLOSED)))$ = set that is created by selecting the first k elements in the set $GS(SEL1(NON\text{-}CLOSED))$ sorted in nondecreasing order according to function $\{q^N\}_i$. A particular case of this strategy is $ST_{d,s,1}$, called the Strategy of Best Operators (Best Search Strategy).

7. Strategy $ST_{d,s,s,k}$ (i.e. the depth-search strategy, with the selection of a node, sorting, and the selection of the k best operators).

   $SEL1_{d,s,s,k}(NON\text{-}CLOSED)$ = a node of minimum value of function QF among all nodes that are created as the extension of the recently expanded node (not necessarily of the recently opened node).

   $SEL2_{d,s,s,k}$ = similarly to $SEL2_{d,s,k}$.

   Similarly, one can define "k-children" strategies $ST_{QF,k}$, $ST_{CF,k}$, $ST_{d,k}$.

8. Strategy ST_{RS} of Random Search.

   $SEL1_{RS}(NON\text{-}CLOSED)$ = randomly selected node from NON-CLOSED.

   $SEL2_{RS}\ (GS(SEL1_{RS}\ (NON\text{-}CLOSED)))$ = randomly selected subset of descriptors.

9. Strategy $ST_{RS,d}$ of Random Search Depth.

   $SEL1_{RS,d}(NON\text{-}CLOSED)$ = recently opened node from *NON-CLOSED*.

   $SEL2_{RS}\ (GS(SEL1_{RS,d}\ (OPEN)))$ = randomly selected descriptor.

Similarly, one can specify many other strategies by combining functions *SEL1* and *SEL* given above.

Let us now introduce few measures of quality of strategies.

$k_1 = CARD(B_a)$, where B_a is the set of all closed and semi-open nodes that were created until all minimal solutions have been found.

$k_2 = CARD(Bs)$, where B_s is the set of all closed and semi-open nodes that were created until one minimal solution has been found.

$k_3 = CARD(V_a)$, where $V_a \supseteq B_a$ is the set of all closed, semi-open, and open nodes that were created until all minimal solutions have been found.

$k_4 = CARD(V_s)$, where $V_s \supseteq B_a$ is the set of all closed, semi-open, and open nodes that were created until one minimal solution has been found.

$k_5 = CARD(T_a)$, where $T_a \supseteq B_a$ is the set of nodes that were created until proving the minimality of solutions, it means the total number of nodes that have been created by a strategy that searches all the minimal solutions.

$k_6 = CARD(T_s)$, similarly to k_5, but for a strategy that searches a single solution.

$k_7 =$ max $SD(N_i)$ - the length of the maximal path (branch) in the tree.

The advantage of the ordered search strategy is the relatively small total number of generated nodes (coefficients k_5 and k_6). The following theorem is true, similar to the theorem from Nilsson [Nilsson71].

Theorem 6.10.1. *If QF satisfies equations* 6.4.8, 6.4.9, 6.4.10, 6.4.11, 6.4.12 *and the ordered search strategy has been chosen (i.e. the strategy A* of Nilsson is being realized) and when some solution of cost QF' has been found, such that all nodes of costs smaller than QF' have been closed, then this solution is the exact minimal solution.*

It is important to find conditions, for which this algorithm finds the optimal solution, generating relatively few nodes. The theorem below points to the fundamental role of function $\hat{h}$. The way in which function $\hat{h}$ is calculated, can substantially influence the quality of solutions in approximate version, or efficiency of the algorithm in exact version.

Let ST_1 and ST_2 be two A^* Nilsson strategies, and $\hat{h}_1$ and $\hat{h}_2$ their heuristic functions. We will define that strategy ST_2 is *not worse specified* than strategy ST_2 when for all nodes N it holds:

$$h(N) \geq \hat{h}_1(N) \geq \hat{h}_2(N) \geq 0 \qquad \textit{Equation 6.10.3}$$

which means, both functions evaluate h from the bottom, but function $\hat{h}_1$ does it more precisely than $\hat{h}_2$.

Theorem 6.10.2. *If ST_1 and ST_2 are A^* Nilsson strategies, and ST_1 is not worse specified than ST_2, then, for each solution space, the set of nodes closed by ST_1 (before the minimal solution is found) is equal to the set of closed nodes of ST_2, or is included in it.*

This theorem says, in other words, that if we limit ourselves to A^* Nilsson strategies only, then there exists one strategy, not worse than all remaining strategies, since it closes not more nodes of the tree than any other strategy. This is the strategy that most precisely evaluates the function **h**, preserving of the equations 6.4.9, 6.4.10, 6.4.11.

For many classes of problems the ordered search strategy is very inefficient because it generates its first solution only when very many nodes have already been created. Next it proves its optimality relatively quickly. In cases, when the user wants to find quickly some good solution, but the **exactness** of the solution is only of secondary importance, it is better to use one of the variants of the branch-and-bound strategies that search in depth.

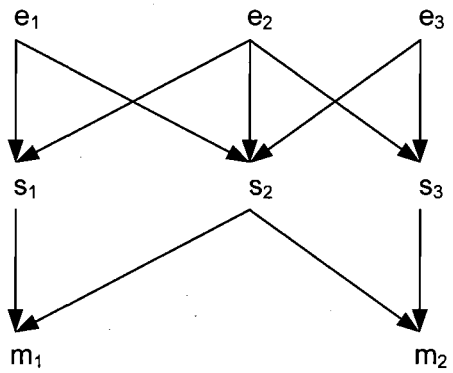


Figure 6.10.1: The example of the lattice with three maximum and two minimum elements. Arrows show the partial order relation.

6.10.1. Properties of branch-and-bound strategy.

Many properties can be proven for the branch-and-bound strategy presented above. We assume that

for each *N, NN, QF(N)* $\neq$ *QF(NN)* and *QF(NN)* > *QF(N)* for *NN* $\in$ *SUCCESSORS(N)* *Equation 6.10.1.1*

6.10.1.1. The branch-and-bound strategy is convergent, independent on function *QF*. Also the specific strategies included in it (such as "depth-first",

"ordered search", etc,) are therefore convergent as well, if the user has not declared some additional cut-off conditions (that may cause the loss of the optimal solution). Some of these strategies do not require calculating function *QF* satisfying certain conditions. This property is an advantage of the given above universal search strategy, when compared with the *A** Nilsson Strategy.

6.10.1.2. If the user is able to define the quality function *QF** such that

$$(\forall N_i, N_j)[QF^*(N_i) < QF^*(N_j) \Rightarrow f(N_1) \leq f(N_j)], \qquad \textit{Equation 6.10.1.2}$$

then the strategy ST_{QF} is optimal in the sense of the number of opened nodes. Only the nodes that are on the paths leading to solutions are extended (other nodes are also opened).

6.10.1.3. If additionally the user succeeds to find a quality function for operators q^{*N} that is **consistent** with *f*,

$$(\forall N, O_1, O_2)[q^{*N}(O_1) > q^{*N}(O_2) \Rightarrow f(O_1) \leq f(O_2(N))], \qquad \textit{Equation 6.10.1.3}$$

then the strategy is optimal in the sense of the number of generated nodes. Only those nodes are expanded, that lay on those paths that lead to minimal solutions. In addition, no other nodes are opened (this concerns the 1-child strategies).

6.10.1.4. It is possible to introduce the relation of partial ordering << on the set of all possible strategies ST_{QF}. It can be proven that the strategies that are *adjacent* in the sense of this order have also similar behavior:

$$if \quad ST_{QF1} << ST_{QF2} \quad then \; k^1 \leq k_i^1 \qquad for \; i = 1,2,...,6.$$

Equation 6.10.1.4

6.10.1.5. The best strategy with respect to relation << (the minimal element of the lattice), is the strategy ST_{QF*}. The "adjacent" strategies are defined. Next it can be proven, that if $QF_0, QF_1, ...,QF_q$ is a sequence of such adjacent functions, then the corresponding strategies, $ST_{QF_0}, ST_{QF_1}, ..., ST_{QF_q}$, are adjacent in the lattice of strategies. Therefore, in the class of the ordered search strategies function *ST* is in a sense a continuous function of function *QF*: small changes of *QF* cause small changes of ST_{QF}. If $QF \approx QF*$ then behavior of ST_{QF} is close to optimal. If the user is able to make choices among **all** functions *QF*, then by the way of successive experimental modifications he can approach the ST_{QF*} strategy.

6.10.1.6. Since strategies "depth-first" are very sparsely located in the lattice (they have high distances from one another), small changes of *QF* can cause a "jump" from ST_{QF*} to a lattice element that is located far from it. Similarly, small modification of *QF* in the direction of *QF** do not necessarily lead to the improvement of the algorithm's behavior.

6.10.1.7. It can be shown, that in the sense of some of the measures introduced above, the proposed algorithm is better than the branch-and-bound algorithms investigated by Ibaraki [Ibaraki76].

Usually, the user should always try to find function QF close to QF^*. With better functions **QF**, the program will find good solutions sooner, where by good solution we understand those with small values of CF_{min} (the decrease of coefficients k_1 - k_4). Therefore, the cut-off of subsequent branches will be done with a smaller value, which will in turn decrease the values of k_5 and k_6. When the depth-first strategy is selected, the changes in behavior can occur in jumps. In addition, with respect to 3), the user has to select function q. With respect to equations 6.8.3 - 6.8.12, respectively, he has to define relations on descriptors and states.

When constructing the strategies, the user has also to keep in mind the following.

6.10.1.8. Generally, for those branch-and-bound strategies that search in depth, it is necessary to define that every branch of the tree terminates with a solution found. In addition the branch is determined with certain constructive conditions of cutting-off (for instance, the cutting-off occurs when certain depth of the tree was reached, or when there are no more operators to apply). The lack of these conditions may lead to the danger of infinite depth-search, or a very long depth-search. For instance, in case of strategy $ST_{d,1}$. This condition is not necessary for A^* Nilsson strategy, which is a special case of the strategy.

6.10.1.9. With respect to parameters k_1 - k_4, the strategies that combine properties of strategies $ST_{d,s,s,k}$, A^* Nilsson Strategy, and $ST_{d,s,k}$, have the best performance.

6.10.1.10. With respect to parameter k_7 the 1-child strategies are the best, and the $ST_{d,s,1}$, strategy in particular.

6.10.1.11. When the user looks for a solution with the minimal depth in the tree, the breadth-search strategy creates theoretically the exact solution as the first solution generated, which is sometimes good. However, the tree can grow often so rapidly, that the strategy cannot be used. Yet in other problems, it is good to use the disk memory. The strategy is useful when the problem is small, or when one can define powerful relations on descriptors or relations on states of the search space.

6.10.1.12. When the depth is limited or when good upper bounds can be found, the depth-first strategies allow to find the solutions faster. Depth-first strategies are good when there are many solutions. They are memory efficient. These strategies are not recommended when the cost function does not increase monotonically along the branches, allowing thus to use the cutting-off.

6.10.1.13. Strategies ST_{QF}, and $ST_{d,s,k}$, often require the shortest times of calculations. The second strategy requires a smaller memory.

6.10.1.14. By constructing strategies that use quality functions one has to take into account that the evaluation of a more complex function allows to decrease the search. It takes, however, more time. Therefore, the trade-offs must be experimentally compared.

6.10.1.15. It is possible to combine all presented strategies, and also to add new problem-specific properties to the strategies. The user can, for instance, create from the depth-first strategy and breadth-first strategy a new strategy that will modify itself while searching the tree, and according to the intermediate solutions found. Another useful trick is to cut-off with some heuristic values, for instance some medium value of CF_{min} and $CF_{min\ min}$.

6.10.1.16. An advantage of random strategies is a dramatic limitation of required space and time. These strategies are good, when used to generate many good starting points for other strategies, and these other strategies find next the locally optimum solutions.

6.11. Switch Strategies

6.11.1. Principles

There are two types of Switch Strategies:

- switch strategies for branches,
- switch strategies for trees.

Below, we will present them both.

A Switch Strategy is defined by using the conditional expression:

$$[\ sc_1 \rightarrow (MM_1, TREE_1), \ldots\ldots, sc_n \rightarrow (MM_n, TREE_n)\ ] \quad \textit{Equation 6.11.1.1}$$

where

1. sc_1,....., sc_n are *switch conditions*,

2. $MM_i = (M_i, ST^i)$ are methods to solve problems by Universal Strategy,

3. M_i are tree methods,

4. ST^i are pure strategies,

5. $TREE_i$ are initial trees of methods MM_i (trees after strategy switchings).

The meaning of formula 6.11.1.1 is the following. If condition sc_1 is satisfied, then use method MM_1 with initial tree $TREE_1$. Else, if condition sc_2 is satisfied, then use method MM_1 with initial tree $TREE_2$. And so on, until sc_1 is encountered.

In practice, M_i, ST^i and $TREE_i$ are defined by certain *changes* to the actual data. These can be some symbolic transformations, or numeric transformation. They can be also the selections of new data structures. Therefore one has to declare the initial data: M_0, ST^0 and $TREE_0$.

- In Switch Strategy for a Branch, the conditions sc_i i, $i = 1$,..., n are verified when a new node is created. These conditions can be also verified in one of the following cases:

  (1) a new node being a solution is created,

  (2) a node is found, being a solution better than the previous solution. The type of the node is specified by the parameters.

- In the Tree Switch Strategy, the conditions are checked after a full tree search of some type has been completed.

In both types of strategies, the conditions of switching strategies can be defined on:

- nodes *NN*,
- branches leading from N_0 to NN,
- expanded trees.

There can exist various *Mixed Strategies* STM, defined as follows

$$STM = (SST_T, SST_B) \qquad \textit{Equation 6.11.1.2}$$

where

$\mathbf{SST_T}$ - is a Tree Switch Strategy,

$\mathbf{SST_B}$ - is a Switch Strategy for a Branch.

For both the Switching Strategies for Tree, and Switching Strategies for Branches, there exist eight possible methods of selecting changes. These methods are specified by one of the subsets of the set $< M_i, ST^i, TREE_i >$ In a special case, by selecting an empty set, changes of M_i, ST_i or $TREE_i$ are not specified. This corresponds to a pure strategy ST_0 (which was declared as the first one). Pure strategies are therefore a special case of the switch strategies.

Similarly, complex methods, defined as $CM = (M_1,....,M_r, STM)$ are generalizations of methods MM_i.

Changes of $TREE_i$, M_i, and ST_i will be now presented.

1. The following changes of $TREE_i$ has been considered.

- change of coordinates of nodes (locally, or in a branch, or in the whole tree),
- adding or removing some coordinates (locally, or in a branch, or in the whole tree),
- cut-off the tree.

2. Changes of M_i by use of a switch strategy can be executed by specifying new components of the solution space. The strategy for Graph Coloring from new text found in chapter is an example of a switching strategy that changes both *TREE* and *M*.

3. Strategy is modified by determining the Change of Strategy Parameters. For instance, the modification of the strategy consists in:

   (1) a permutation of list *OPEN*,

   (2) a selection of some its subset,

   (3) some modification to list *OPERT*.

   Since the entire information about the solution tree is stored in list *OPEN*, the new strategy can start working immediately after the Branch Switch. The Main Universal Search Subroutine is constructed in such a way, that even by applying the switch search strategy it is still possible to obtain the exact solution.

6.11.2. Examples of Switch Strategies

6.11.2.1. The Far-Jumps Strategy. This strategy finds solutions with high mutual distances in the solution space. At first, the Breadth-First Strategy with macro operators and dominance relations is used to develop a partial tree. Together with each node *N* of the tree also its level in the tree, *SD(N)*, is stored. A node from *OPEN* that has the smallest level is selected. Next the "depth-first" strategy is used until the first solution is found. The program evaluates, using some additional method, whether this is a minimum solution, or a satisfactory solution. When program evaluates that this was not the minimum solution, the "strategy switch" is executed. The strategy switch is executed as follows.

(1) the node with the lowest level in the actual list *OPEN* is selected;

(2) this node is added at the beginning of list *OPEN*. Starting from this node, the tree is expanded again using the depth-first strategy, until the next solution is found, etc. With each solution, the order of nodes in *OPEN* can be modified.

6.11.2.2. The Distance Strategy. An advantage of this switch strategy is that the successively generated solutions are placed far away one from another. This gives the possibility of "sampling" in many parts of the space, which can lead to quicker finding of good cut-off values (this happens thanks to the jumping-out of the local minima of the quality function). It may be useful, that the "sampling" property is the opposite to the "depth-first" or other pure search strategies.

6.11.2.3. The Strategy of Best Descriptors. The principle of this strategy is that it stores, for some pure strategy (for instance the depth-first, or the ordered-search), all the descriptors that proved to be the most useful in finding the previous solution. Sometimes, only some of these descriptors are stored. For instance, the dominating descriptors, or the descriptors with the highest values of cost or quality functions are stored. After switch, these descriptors are placed at the beginning of list *OPERT*, and are therefore used as the first ones in the next tree expansion. The switch strategies of this type can be applied to find quickly good cut-off values in branch-and-bound strategies.

6.11.2.4. Strategy of Sequence of Trees. This is an example of a strategy that switches trees. It expands some full tree, or a tree limited by some global parameters (time, number of nodes). Next, using some additional principles, it selects few nodes, *SEL_NODES*, of the expanded tree (for instance, the nodes with the minimum value of the cost function). Finally, the strategy expands new trees, each starting from those that start from *SEL_NODES* nodes. It usually uses a different set of components of the space, and/or pure strategy in these new trees. In particular, one of the strategies selects a new set of descriptors. Another strategy of this type, calculates the value of CF_{min} as some function of CF_{min} and other parameters, including the probabilistic evaluations of $CF_{min\ ,\ min}$ during the moment of switching. This strategy is not complete, but it can substantially limit the search by backtracking from smaller depth values.

6.12. Standard versus Quantum Searches.

The methods discussed in sections 6.1 – 6.11 are general and applicable to any parallel processor. For instance, in several problems the best bound search can be realized using repeatedly a single (quantum) Grover processor with oracles modified at every search run.

Example 6.12.1:

Let us analyze for example the PPRM minimization for an incompletely specified function from Figure 6.12.1a. The first quantum search is extended among 2^n positive polarity groups (groups being all products of variables and a group "1").

The positive polarity groups are represented in the Positive Polarity Exor Map from Figure 6.12.1b. Each of these groups (product terms) is realized by one cell of this map. The quantum oracle evaluates the quality function to be maximized being the ratio of ones to zeros in each group. The group c from Figure 6.12.1a is selected as the cheaper one of two groups with the same ratio (circled on top right in Figure 6.12.1b). This selection is done using the first run of Grover with the oracle. After exoring the group c, Figure 6.12.2, the second call to Grover is extended which returns the group ab with ratio 2/0. Exoring this group from function from Figure 6.12.2b creates a function "0" (Figure 6.12.2b) so the search is completed. A general search pattern for this kind of "sequential quantum algorithms" is presented in Figure 6.12.3.

Although this particular example is trivial, it illustrates well the principle of parallel search that uses Grover-based quantum computers.

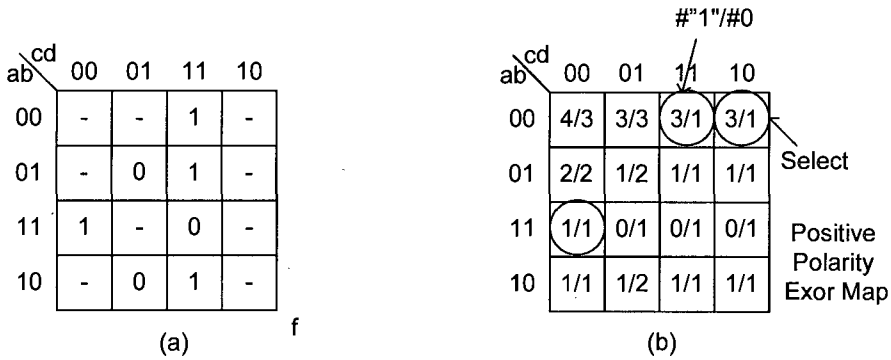


Figure 6.12.1: (a) Incomplete function to be realized as a PPRM, (b) Positive Polarity Exor Map with costs of product terms.

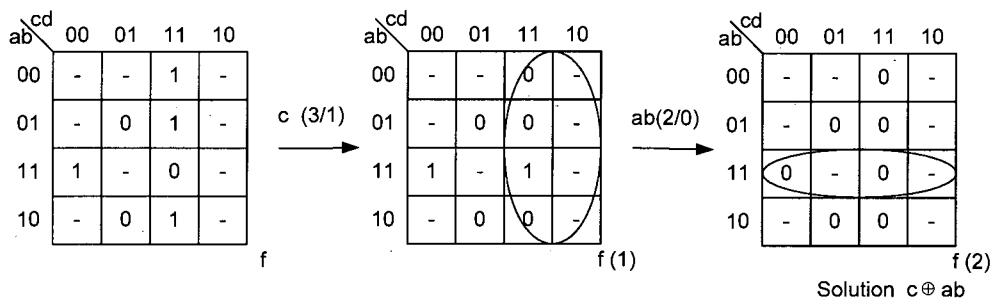


Figure 6.12.2: Exhaustive/greedy strategy based on repeated calls of quantum Grover Algorithm. (a) The original incomplete function to be minimized as a PPRM circuit. (b) the function f (1) to be realized after exoring the best group c with one-to-zero ratio 3/1, (c) the function f (2) to be realized after exoring the best group ab with ratio 2/0 selected in the second call of Grover. As this function f(2) is "0" the search is completed and solution $f = c \oplus ab$ is returned as the best PPRM for f from Figure 6.12.1a. Of course, only one branch of the search tree is shown here for simplification.

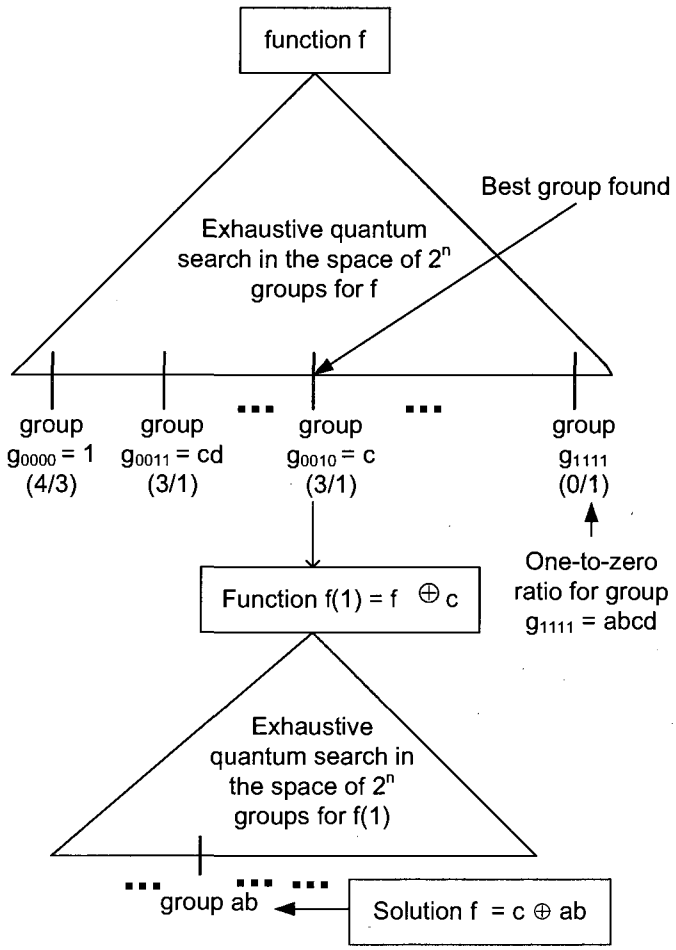


Figure 6.12.3: Visualization of search space of an exhaustive/greedy strategy extended by sequential calls to the quantum Grover accelerator. At each stage Grover Algorithm Accelerator is called to execute exhaustive search of the best product term to be chosen. This visualization illustrates the search from Figures 6.12.1 and 6.12.2.

The multi-strategy search algorithm can be applied to both classical and quantum computing. For instance the heuristics to find a good lower or upper bound in graph coloring are useful in all of the following: classical software, Grover oracle construction and in a hybrid hierarchical parallel search system. In case of classical

search the set of descriptors is at the beginning equal to the number of nodes N. When a solution with k < N nodes is found the search is repeated with only k color descriptors and possibly other strategy is chosen. The same principle is used in quantum search. The Optimizing Oracle assuming N colors would be in most cases very wasteful, so we run a Decision oracle with few iterations probabilistically. The repeated (or parallel) measurements after few Grover Loop iterations will find some solution candidates which are verified on classical computers. This way a good upper bound k colors is found that is next used to construct a smaller oracle. Similarly, finding a maximum clique of a graph can be used to find the lower bound of a chromatic number and next run Grover from it increasing the number of expected colors (see chapter 12).

Example 6.12.2:

Figure 6.12.4 presents application of tree searching for ESOP minimization with "more ones than zeros" heuristics to an incompletely specified function. The function is different than in the previous Example 6.12.1. The single literal groups are: a, $\overline{a}$, b, $\overline{b}$, c, $\overline{c}$, d, $\overline{d}$. The 1/0 ratios for these groups are the following:

a - 2/2, $\overline{a}$ - 2/2, b - 2/1, $\overline{b}$ - 2/3, c – 2/3, $\overline{c}$ - 2/1, d – 4/2, $\overline{d}$ - 0/2.

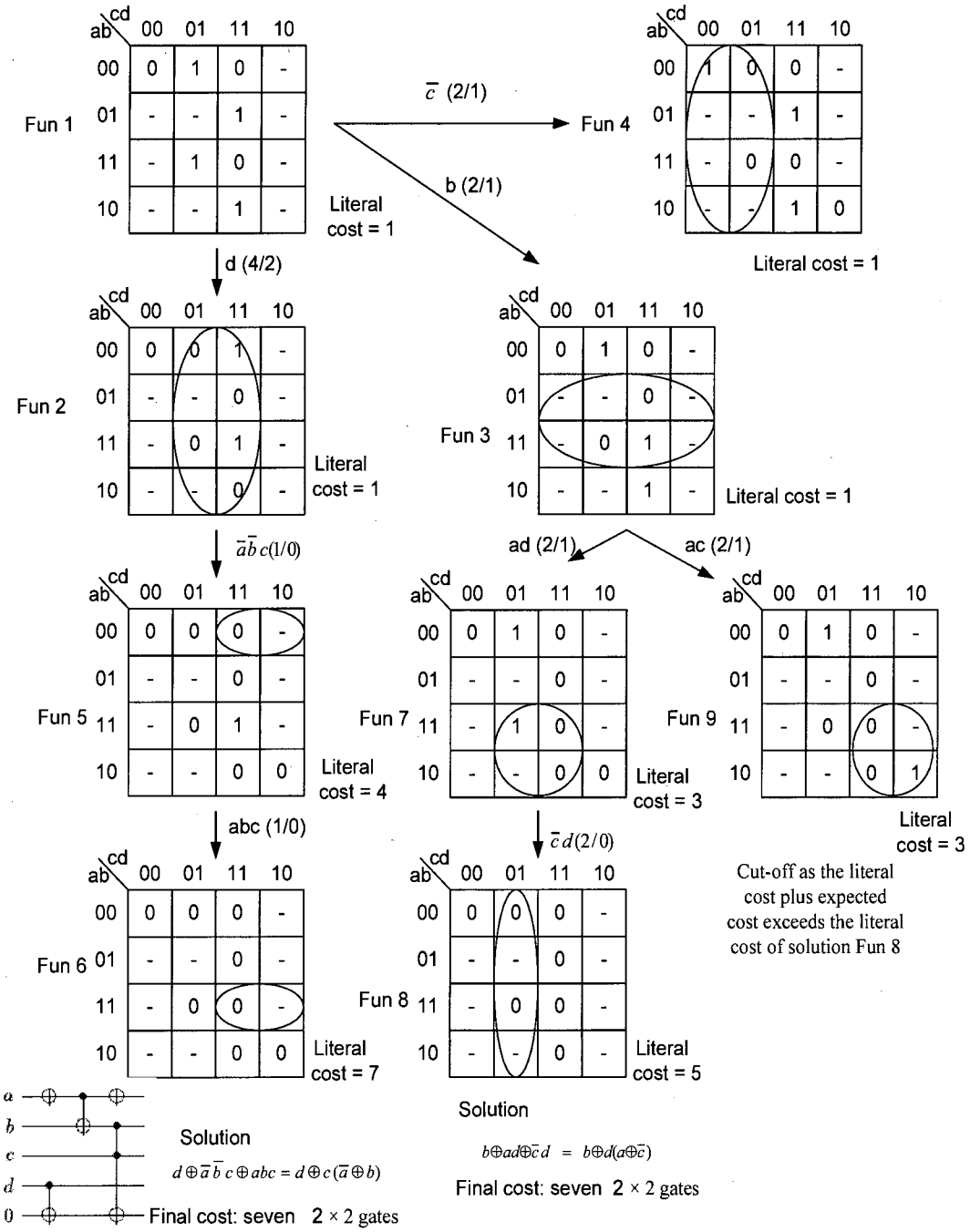


Figure 6.12.4: ESOP minimization search for an incomplete function Fun 1(a, b, c, d). This search is based on "more-ones-than-zeros" heuristics, which can however lead to various subtrees and different hybrid quantum strategies. We recall that the 3×3 Toffoli gate costs five 2×2 gates. This is how the final costs are calculated.

Thus groups b, $\overline{c}$ and d are evaluated as the best choices, as reflected in the first level of search from Figure 6.12.4. Now this search is done exhaustively on a parallel quantum processor from Figure 6.12.5. From function Fun1 the functions are created: Fun2 by exoring group d, Fun3 by exoring b and Fun4 by exoring $\overline{c}$. Out of these functions Fun2 has 2 true minterms while Fun3 and Fun4 have 3 true minterms each. Nodes Fun 2, Fun 3 and Fun 4 are added to the OPEN List of the Master Serial Processor. Node Fun2 of the tree is therefore selected for expansion by Best Bound Tree Search Algorithm as it has the smallest value of the evaluation function. Now the two Slave Processors with quantum co-processors are used to execute parallel quantum search. One is allocated the node Fun 2 and another is allocated the node Fun 3. Node Fun 4 remains in list Open in Master for future expansion. Using the method as in the previous example 6.12.1 the first quantum processor finds the solution $d \oplus c(\overline{a} \oplus b)$ with the cost of seven 2 × 2 gates and the second processor finds solution $b \oplus d(a \oplus \overline{c})$ with the same cost. After backtrack in the standard processor of second quantum processor (the Slave Processor 2) function Fun 9 is found which has a literal cost of 3. But as the groups to cover minterms have at least Literal cost 3 each, 3 + 3 = 6 > 5 which was a literal cost of the solution from node Fun 8. This search branch is thus cutted-off. As both Slave Processors are now finished the Master decomposes Fun 4 and allocates new tasks (not shown) to both Slaves. The process goes on until the final solution is found.

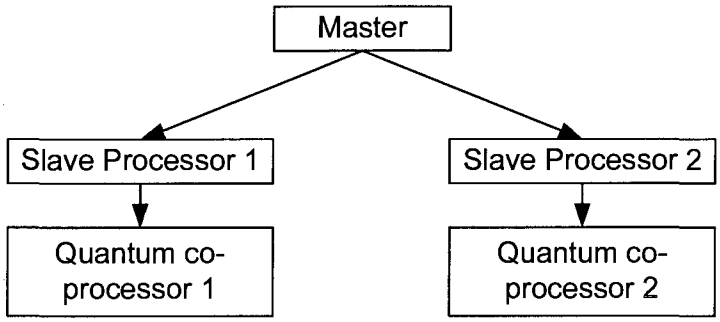


Figure 6.12.5: Master Slave Processor with quantum co-processors used in Example 6.12.2.

Observe that this search method can be applied to PPRM, KRM, GRM, affine extensions, etc. Virtually every problem from this dissertation can be solved like this. Observe also that this method is heuristic, because it uses approximate quality functions and incomplete search in Master. This method can be improved in many ways, using analysis and methods as discussed in sections 6.12 – 6.16 and next chapters. For instance, the search strategy from Example 6.12.2 can be improved by adding a special method to analyze linear variables. This is illustrated below.

Theorem 6.12.1. Function can be represented in the form: f (a, b, c, d) = a $\oplus$ g(b, c, d) iff f $\oplus$ a does not depend on a, i.e.

$\frac{\partial g}{\partial a} = 0$ *or* $g_{\bar{a}} \oplus g_a = 0$. In such case variable a is called the linear variable of function f. It is always worthy to extract first all linear variables from the function that is minimized and next perform the search. This linearization applies to every search sub-problem.

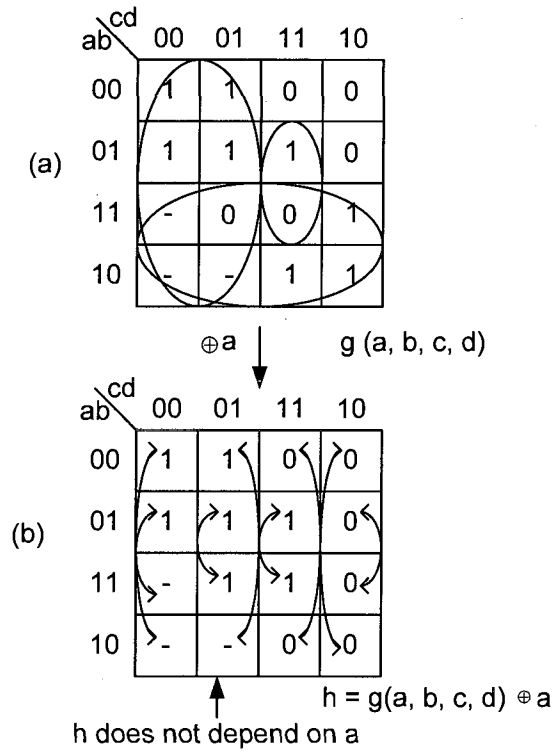


Figure 6.12.6: Verifying if variable a is a Linear Variable of function g(a, b, c, d). (a) the original function g(a, b, c, d), (b) the function h = g(a, b, c, d) ⊕ a. The arrows illustrate the graphical (mirror) verification if $\frac{\partial h}{\partial a} = 0$. *In this case it is so as the minterms on both ends of each arrow are the same. Thus h does not depend on a .*

Example 6.12.3:

Figure 6.12.6a illustrates a function of 4 variables, g (a, b, c, d), to be minimized as ESOP. To check if this function has a linear variable **a** we create function h = g (a, b, c, d) ⊕ a (Figure 6.12.6b). As illustrated in Figure 6.12.6b function h does not depend on variable **a**, thus g (a, b, c, d) = a ⊕ $\hat{h}$ (b, c, d).

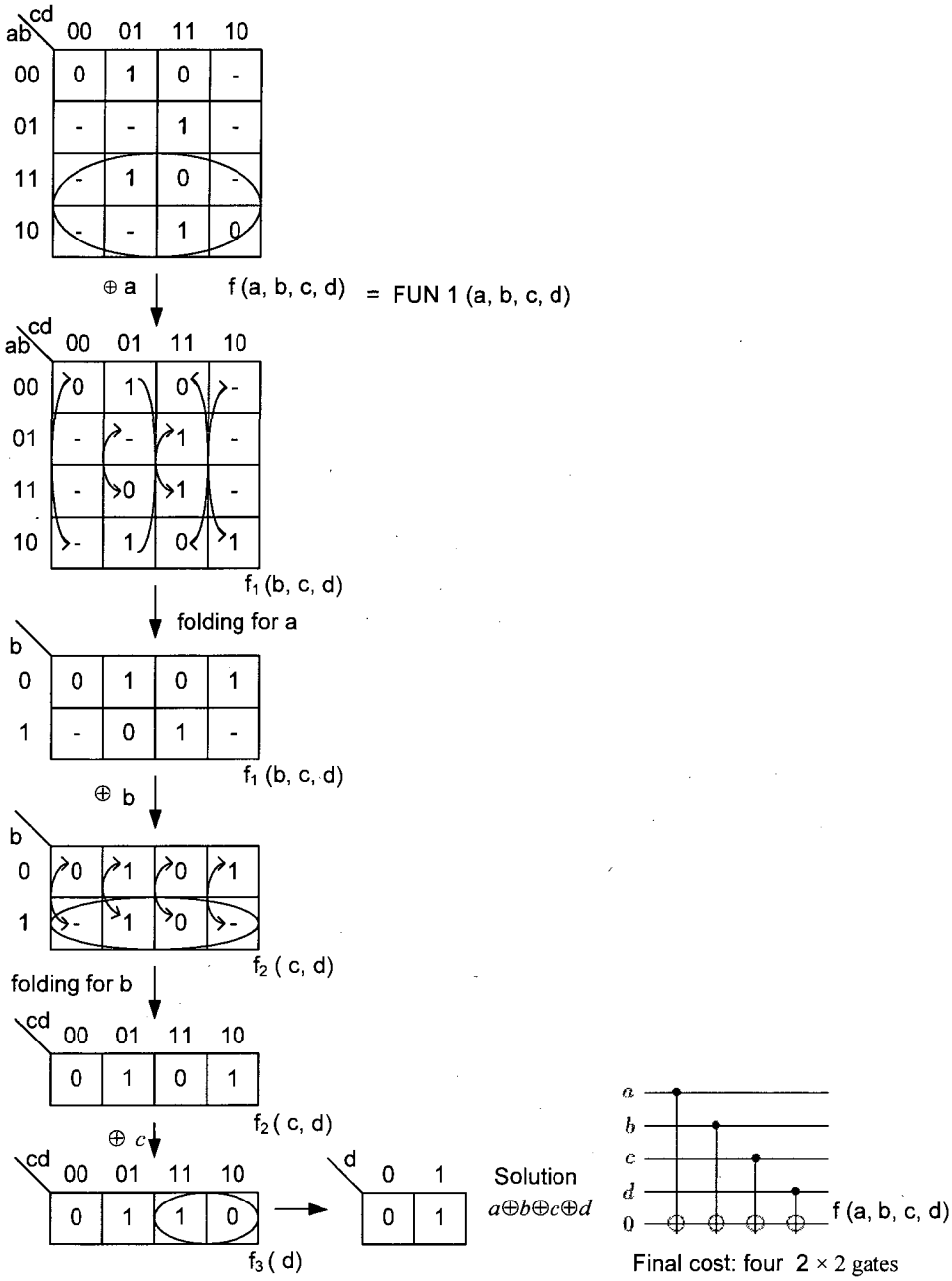


Figure 6.12.7: Tree search with additional linearity test. There is no branching as the function f(a, b, c, d) happens to be linear so the exact solution f (a, b, c, d) = a ⊕ b ⊕ c ⊕ d is found directly by a sequence of extracting linear variables.

Thus variable a is a linear variable of g(a, b, c, d) which should be used in ESOP minimization. We will repeat therefore now the search from previous Example 6.12.2 trying first to extract all linear variables. The process of extracting all linear variables from function Fun 1 from Figure 6.12.4 is presented in Figure 6.12.7. At first it is verified that a is a linear variable of f(a, b, c, d) thus $f_1(b, c, d)$ is created such that f(a, b, c, d) = a $\oplus$ $f_1(b, c, d)$. It is found next by exoring and folding for b that $f_1(b, c, d)$ = b $\oplus$ $f_2(c, d)$. Finally it is found that $f_2(c, d) = c \oplus f_3(d) = c \oplus d$. Thus f(a, b, c, d) = $a \oplus b \oplus c \oplus d$ and the solution is found without any branching, with the final cost of only four 2×2 gates.

This example, together with the previous ones illustrate the power and ease of creating various search strategies using a hybrid hierarchical quantum computer.

6.13. Example of Application: The Covering Problem

The following examples of some partial problems will illustrate the basic ideas involved in the state-space search.The examples will show also the methods that are used to formulate problems for multi-purpose search routines like those proposed in previous sections of this chapter.

6.13.1. The Formulation of the Set Covering Problem

This problem is used in Column Minimization for decomposition. It is also widely encountered in logic design (among others, in PLA minimization, test minimization,

multilevel design - see many recent examples in [Perkowski87]. As an example, let us consider the covering table shown in Figure 6.13.1.

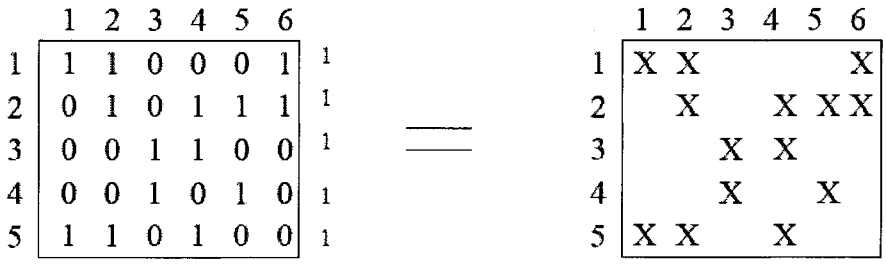


Figure 6.13.1: A Covering Table With Equal Costs of Rows

Each row has its own cost indicated by the value to the right of it. In this example, they are all equal. An **X** at the intersection of row r_i and column c_j means that row r_i covers column c_j. This can be described as:

$$(r_i, c_j) \in COV \subset R \times C, \qquad \textit{Equation 6.13.1.1}$$

or briefly, by $COV(r_i, c_j)$.

A set of rows which together cover all the columns and have a minimal total cost should be found.

The direct problem formulation is as follows:

1. **Given**:

   a. the set $R = \{ r_i , r_i , \ldots , r_i \}$ (each r_i is a row in the table)

b. the set $C = \{ c_1 , c_2 ,..., c_n \}$ (each c_j is a column in the table)

c. the costs of rows $f1 (r_j)$, $j = 1, ... , k$

d. the relation of covering columns by rows is $COV \subset R \times C$.

2. **Find**

Set $SOL \subset R$

3. **Which fulfills the condition:**

$$(\forall c_j \in C)(\exists r_i \in SOL[COV(r_i,c_j)] \qquad \textit{Equation 6.13.1.2}$$

4. **And minimizes the cost function**

$$f2 = \sum_{r_i \in SOL} f1(r_i) \qquad \textit{Equation 6.13.1.3}$$

It results from the above formulation that the state-space $S = 2^R$. This means that $SOL \subset R$. Hence, it results from the problem formulation that all the subsets of a set are being sought. Then, according to the methodology, the standard generator, called T_1, that generates all the subsets of a set is selected. Operation of this generator can be illustrated by a tree.

The previously mentioned relation RE on the set $S \times S$ can be found for this problem and used to reduce searching for a respective search method. It can be defined as follows:

$$s_1 \; RE \; s_2 \Leftrightarrow \; s_2 \supset s_1 \qquad \textit{Equation 6.13.1.4}$$

Therefore, when a solution is found, a cut-off occurs in the respective branch.

There exists for each element $c_j \in C$ an element $r_i \in SOL$, such that their relation COV is met. In other words, r_i covers c_j which means that the predicate $COV(r_i, c_j)$ is satisfied. The cost function F assigns the cost to each solution. In this case, this means that $F = f2$ is the total sum of $f1(r_i)$\$; the costs of rows r_i that are included in set SOL. Thus, using the problem definition from section 6.2, the covering problem is formulated as the problem

$$P = (2^R, \{p_1\}, f2), \qquad \textit{Equation 6.13.1.5}$$

where

$$p_1(SOL) = (\forall c_j \in C)(\exists r_i \in SOL[COV(r_i, c_j)] \qquad \textit{Equation 6.13.1.6}$$

In case of classical search this problem was formulated using logic equations, Lists or binary matrices. In case of quantum search the most natural is to have variables corresponding to rows of the table, but it still gives freedom in oracle construction.

6.13.2. Tree Search Method 1

The initial tree search method based on the direct problem formulation is then the following

1. The initial node $N_0 : (QS, GS, F) := (\emptyset, R, 0)$.
2. The descriptors are rows r_i . The application of the operator is then specified by the subroutine $O(N, r_i) =$

$$
\begin{aligned}
[\ & GS(NN) := GS(N) \setminus \{ r_i \} \\
& QS(NN) := QS(N) \cup \{ r_i \} \\
& CF(NN) := CF(N) + f_1 (r_i) \\
]
\end{aligned}
$$

3. Solution Problem and Condition (cut-off type)

$$p_1(NN) = (\forall c_j \in C)(\exists r_i \in QS(NN)[COV(r_i, c_j)] \qquad \textit{Equation 6.13.2.1}$$

Comments.

1. *NN* denotes a successor of node *N*.
2. *QS(N)* is the set of rows selected as the subset of the solution in node *N*.
3. *F(N)* is the cost function for node *N*. This is the total sum of costs of the selected rows from *QS(N)*.

As we can see in this problem, the formulation of the additive cost function is possible.

An example of the cover table is shown in Figure 6.13.1. In this example, to simplify calculations, we assumed equal cost of rows. However, the method can be easily extended to arbitrary costs of rows. The solution tree obtained from such a formulation is shown in Figure 6.13.2.

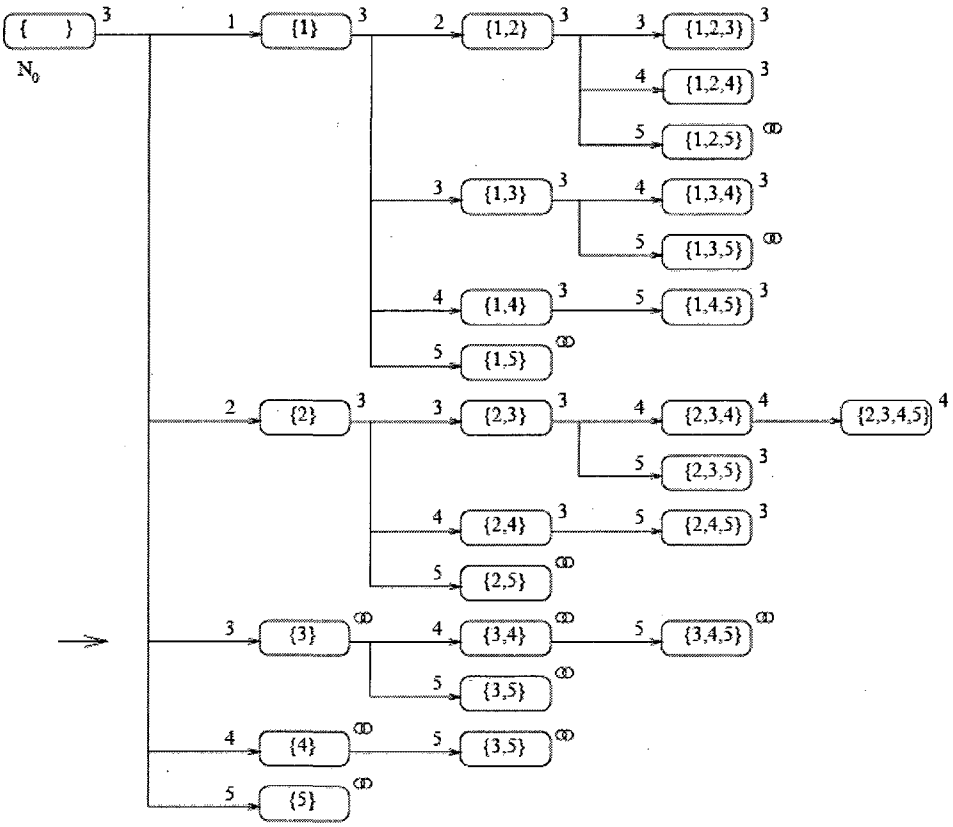


Figure 6.13.2: First Search Method for the Table from Figure 6.13.1.

This method is the simplest and the most natural for quantum oracles. Remember oracle for graph coloring and SAT. But it is not much knowledge-based and thus expensive. It can be used however in each quantum processor to deal with intelligently decomposed problems.

The nodes of the search tree are in the ovals. The arrows correspond to the applications of operators, and each descriptor of operator stands near the corresponding arrow. The solution nodes are shown in bold ovals. The costs of nodes are outside the ovals, to the right. The sets inside the ovals correspond to partial solutions in the nodes. Since the entire tree has been developed here, the sets GS for each node can be reconstructed as the sets of all descriptors from the outpointing arrows.

The cutting-off uses the fact that the cost function increases monotonically along the branches of the tree; this is the cut-off condition. The nodes that are the solutions are therefore not extended. If the cut-off conditions were not defined, for example, the nodes {1, 2, 3} and {1, 2, 4} would be extended. Otherwise, the tree is produced under the assumption that the cutting-off is not done for the solutions with cost function values worse than for those nodes previously calculated. The values of function f for nodes are shown to the right of these nodes.

Observe that some nodes of the tree are created (for example, node {3}) in a way that does not allow any solutions to be produced in their successor nodes. Because each column must be covered by at least one row in the node, the generation of such nodes can be avoided. This is done by storing the columns c_j that are not yet covered in set AS. The branching for all rows r_i that cover the respective column for each

individual column is also generated. These are such rows r_i that $COV(r_i , c_j)$. We can now formulate a new tree search method

6.13.3. Tree Search Method 2

1. Initial node N_0

$$(QS,\ GS,\ AS,\ CF) := (\emptyset, \{ r_k \in R \mid COV(r_k , c_1) \}, C, 0) \qquad \textit{Equation 6.13.3.1}$$

The first element of C is denoted by c_1 above.

2. Operator

$O(N, r_i) = [$

$QS(NN) := QS(N) \cup \{ r_i \}$

$AS(NN) := AS(N) \setminus \{ c_j \setminus \in C \mid COV(r_i , c_j)\}$

c_j := the first element of $AS(NN)$

$GS(NN) := \{ r_k \in R \mid COV(r_k , c_j)\}$

$CF(NN) := CF(N) + f_1 (r_i)$

$]$

3. Solution condition (cut-off type)

$$p_1(NN) = (AS(NN) = \emptyset) \qquad \textit{Equation 6.13.3.2}$$

The corresponding tree is shown in Figure 6.13.3.1.

Two disadvantages to this method become apparent from Figure 6.13.3.1. The first disadvantage is creating the redundant descriptor 4 in $GS(N_5)$. This descriptor cannot be better than the descriptor 2. This disadvantage can easily be overcome by writing a new code for this section, that would define and use the domination relation on descriptors. The second disadvantage is due to the repeated generation of the solution {1, 3, 4}, the second time as {1, 4, 3}. If the optimal solution is desired, then there is no way to avoid the inefficiency introduced by the Tree Search Method 2.

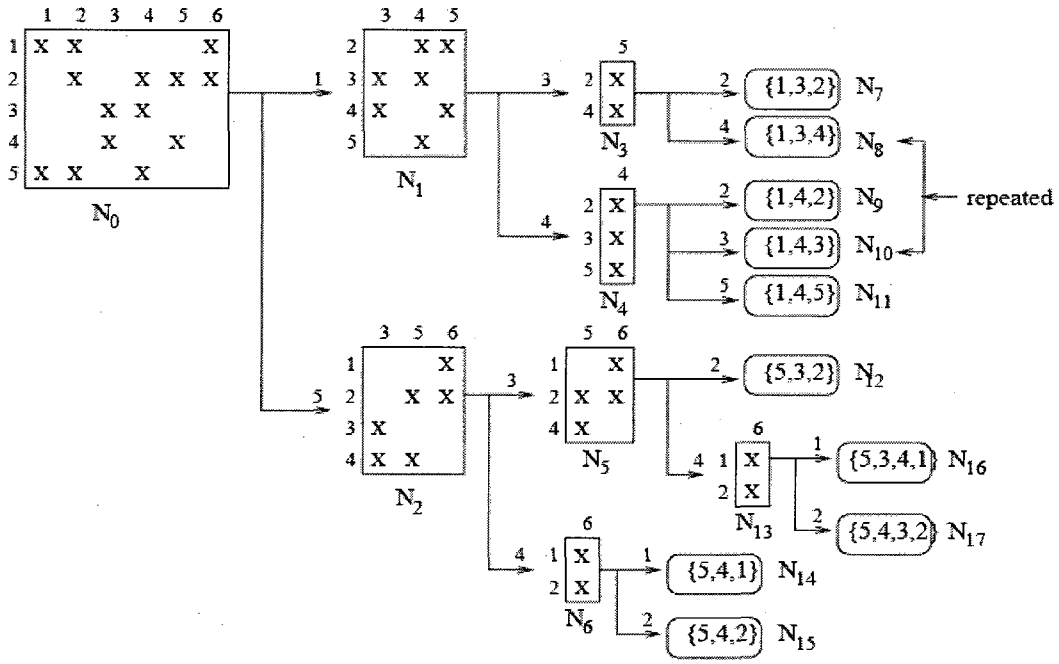


Figure 6.13.3.1: Second Search Method for the Table from Figure 6.13.1.

This method is good for a Master Processor that decomposes a problem to smaller problems and sends these smaller problems to Slave Quantum Processor. Assume as an example that at most 4 × 4 matrices, or smaller, can be handled by a quantum processor. Then the initial matrix from Figure 6.13.3.1 can not be handled but each of smaller matrices after initial decomposition can be handled and solved in parallel on 2 quantum processors. Of course, the example is trivial and does not require quantum search, it serves only the concept explanation.

6.13.4. Tree Search Method 3

Another method to avoid generating nodes for which $f = \infty$ is the application of the first method (the generation of the T_1 type of tree) and an additional filtering subroutine to check nodes to verify if the set of rows from *GS(N)* covers all the columns from *AS(N)*. In addition, the following code of type *"Actions on the Selected Node"* is created:

If

$$AS(N) \not\subset \{c_j \in C)(\exists r_i \in GS(N))[COV(r_i, c_j)] \qquad \textit{Equation 6.13.4.1}$$

then

$GS(N) := \emptyset$

This means, that the cut-off is done by clearing set *GS(N)* when the set of all the columns covered by the available descriptors from *GS(N)* does not include the set *AS(N)* of columns to be covered. For example, at the moment of generation shown by the arrow in Fig. 6.13.2, the set of $GS(N_0) = \{3,4,5\}$ and it does not cover $AS(N_0) = C$. Therefore, it is assigned $GS(N_0) := \emptyset$, and the generation of the subtree terminates. This forms the Tree Search Method 3.

6.13.5. Tree Search Method 4

The generated tree can be decreased even further when the second method is used and it is declared in the operator that:

$$GS(NN) := \{ r_k \in GS(N) \setminus \{ r_i \} \mid COV(r_k, c_j) \} \qquad \textit{Equation 6.13.5.1}$$

Let us recall that symbol \ denotes operation of set difference.

However, this approach can cause losing the optimal solution. It is then a typical *heuristic directive* and not a *methodic directive* like those discussed previously. In both trees, the cutting off condition based on the cost function has been not yet considered. If the solution {1, 2, 3} in the tree shown in Figure 6.13.2 were first found, node {2, 3, 4} could be cut off, and the non-optimal solution {2,3,4,5} would not be generated. However, until now, only the methods of constructing the generator of complete and non-redundant trees have been presented. These are the trees calculated

for the worst case of certain rules and heuristics that will be discussed in sections 6.13.5 and 6.13.6.

Search Strategies

Various search strategies can be illustrated using this example, to give the reader an intuitive feeling for the concepts and statements introduced in the previous sections. This has application in classical software and serial pre-processing/decomposition in a hybrid quantum system.

The node enumeration order from Figure 6.13.3.1 corresponds to the Breadth-First strategy, and to the strategy of Equal Costs (with respect to the equal cost of rows applied in this example).

Eight nodes were generated in node N_7 to find the optimal solution {1, 3, 2}. The optimality of the solution {5, 4, 2} was proven after creating node N_{15}, which means, after generating 16 nodes. Nodes N_7 to N_{15} were temporary. Cost-related backtracking occurs in node N_{13} and, therefore, nodes N_{16} and N_{17} are not generated.

The strategy Depth-First generates the nodes in the order N_0 , N_1 , N_2 , N_5 , N_6 , N_{14} , N_{15} , N_{12} , N_{13} , N_3 , N_4 , N_9 , N_{10} , N_{11} , N_7 , N_8 . After finding N_{14}, i.e., generating six nodes, the optimal solution {5, 4, 1} is found. As in the previous strategy, after generating 16 nodes, the optimality of the solution {1, 3, 4} is determined. We can state - *"it is proven"*, since the method is exhaustive, and we have generated all nodes.

The strategy Depth-First-With-One-Successor, generates the nodes in the order: N_0 , N_1 , N_3 , N_7 , N_8, N_4 , N_9 , N_{10} , N_{11} , N_2 , N_5 , N_{13} , N_6 , N_{14} , N_{15} . The optimal solution {1, 3, 2} is found after creating four nodes. After generating 16 nodes, the optimality of the solution {5, 4, 2} has been proven. Because the selection of the descriptor depends on the row order among the rows covering the first column, the selection is arbitrary. Hence, in the worst case, the order of generation could be N_0 , N_2 , N_5 , N_{13} , N_{16} (the temporary solution {5, 4, 3, 1} of cost 4 has been found), N_{17} , N_{12} , N_6 , N_{14} , N_{15} , N_1 , N_3 , N_7 , N_8 , N_4 , N_9 , N_{10} , N_{11}. A tree of 18 nodes would be generated to prove the optimality of {1, 4, 5}. This illustrates, that good heuristics are very important to limit the size of the solution tree.

6.13.6. Tree Search Method 5

Subsequent advantages will result from the introduction of the heuristic functions that control the order in which the tree is extended, with regard to the method 2 presented above. The introduction of such functions will not only lead to finding of the optimal solution sooner, but also to expediting the proof of its optimality. This is due to fuller use of the cutting-off property, which results in a search that is less extensive when the optimal solution is found earlier.

The quality function for the operators with regard to the selection of the best descriptors in the branching nodes, as well as the quality function for nodes with regard to the selection of the nodes to be extended is defined below.

Quality function for nodes:

$$QF(NN) = CF(NN) + \hat{h}(NN) \qquad \textit{Equation 6.13.6.1}$$

where

$$\hat{h}(NN) = CARD(AS(NN)) \; . \; CARD(GS(NN)) \; . \; K, \qquad \textit{Equation 6.13.6.2}$$

and

$$K = \frac{\sum_{r_i \in GS(NN)} f_1(r_i).CARD\{c_j \in AS(NN) \mid COV(r_i, c_j)\}}{(\sum_{r_i \in GS(NN)} CARD\{c_j \in AS(NN) \mid COV(r_i, c_j)\})^2} \qquad \textit{Equation 6.13.6.3}$$

Such a defined function $\hat{h}$ is relatively easy to calculate. As proven in the experiments, it yields an accurate evaluation of the real distance h of node NN from the best solution. It is calculated as an additional coordinate of the node's vector. The function's form is an outcome of the developer trying to take into account the following factors:

1. The nodes N_i are extended for which the fewest columns need to be covered in the $AS(N_i)$. There is a higher probability that the solution is in the subtree $D(N_i)$ at the shallow depths for such nodes. Hence, the component *CARD(AS(NN))*.

2. The nodes, for which the fewest decisions need to be made, are extended. This is a general directive of tree searching. It is especially useful when there exist strong relations on descriptors, as happens in our problem. Hence, the component *CARD(GS(NN))*.

3. The coefficient K was selected in such a way that, with respect to the properties of the strategies discussed previously, the function $\hat{h}$ is as near to h as possible.

The quality function for operators is defined by the formula

$$q^{NN}(r_i) = c_1 f_1(r_i) + c_2 f_2(r_i) + c_3 f_3(r_i), \qquad \textit{Equation 6.13.6.4}$$

where c_1, c_2, c_3 are arbitrarily selected weight coefficients of {\em partial heuristic functions} $f_1, f_2,$ and f_3 defined as follows

f_1 has previously been defined as the cost function of rows

Equation 6.13.6.5

$$f_2(r_i) = CARD\ \{ c_j \in AS(NN) \mid COV(r_i, c_j) \} \qquad \textit{Equation 6.13.6.6}$$

$$f_3(r_i) = \frac{1}{f_2(r_i)} \sum_{j=1}^{n} CARD[r_e \mid c_j \in AS(NN) \wedge COV(r_i, c_j) \wedge r_e \in GS(NN) \wedge COV(r_e, c_j)]$$

Equation 6.13.6.7

where n is number of columns. Function $f_3(r_i)$ defines the "resultant usefulness factor of the row" r_i in node *NN*. Let us assume that there exist k rows covering some

column in the set *GS(NN)*. The value of the usefulness factor of each of these rows with respect to this column equals k. When $k = 1$, the descriptor is *indispensable* (or with respect to Boolean minimization, the corresponding prime implicant is *essential*). The *resultant usefulness factor of the row* is the arithmetical average of the *column usefulness factors* with respect to all the columns covered by it. Then, one should add an instruction in the operator subroutine to sort the descriptors in *GS(NN)* according to the non-increasing values of the quality function for descriptors q^{NN}.

The next way of decreasing the solution tree is by declaring new section code that checks the relations on descriptors. If the descriptors r_i and r_j are in the *domination relation* in the node N (such relation is denoted by $r_i > r_j$), r_j can be removed from *GS(N)* with the guarantee that at least one optimal solution will be generated. If the descriptors r_i and r_j are in the *global equivalence relation* in node N, any one of them can be selected. The other descriptor is removed from *GS(N)*, as well as from *GS(M)* where M is any node in the sub-tree *TREE(N)*. The *equivalence class* $[r]$ of some element r from *GS(N)* is replaced in this coordinate by r itself. Descriptors declared as locally equivalent are treated similarly. The only difference is that the descriptors are then removed from *GS(N)* only. Observe, that these relations are not based on costs, but on some additional problem-dependent information about the nodes of the tree that is available to the program. The covering problem may be a good example of this property.

The descriptors r_1 and r_2 are *globally equivalent* in node *NN* when they have the same cost and cover the same columns

$$r_1 \equiv r_2 \Leftrightarrow f_1(r_1) = f_1(r_2) \wedge (\forall \in AS(NN))[COV(r_1,c) = COV(r_2,c)] \quad \textit{Equation 6.13.6.8}$$

Descriptors (rows) r_1 and r_2 are *locally equivalent* in node *NN* when, after removing one of them from the array, the number of columns covered by j rows is the same for each $j = 1,...,$ *CARD(GS(NN))* - 1 as after removing the second one.

$$r_1 \cong r_2 \Leftrightarrow (\forall j = 1,.....,CARD(GS(NN)) - 1)[LK(j,r_1) = LK(j,r_2)]$$

Equation 6.13.6.9

where *LK(j,r)* is the number of columns covered by j rows in the array that originates from *M(NN)* after removing row r.

$$LK(j,r) = CARD\ \{ c_k \in AS(NN) \mid CARD(X_k) = j \} \quad \textit{Equation 6.13.6.10}$$

where X_k is the set of rows covering the column c_k

$$X_k = \{ x \in GS(NN) \setminus \{r\} \mid COV(x, c_k) \} \quad \textit{Equation 6.13.6.11}$$

Descriptor r_1 is *dominated* by descriptor r_2 when: (1) it has larger cost than r_2, and (2) r_1 covers at most the same columns as r_2, or when (3) r_1 has the same cost as r_2, and covers the subset of columns covered by r_2,

$$r_1 \leq r_2 \Leftrightarrow f_1(r_1) = f_1(r_2) \wedge (\forall c_k \in AS(NN))[COV(r_1,c_k) = COV(r_2,c_k)]$$

$$\vee f_1(r_1) = f_1(r_2) \wedge \{c_k \in AS(NN) \mid COV(r_1, c_k) \subset \{c_k \in AS(NN) \mid COV(r_2, c_k)\}$$

Equation 6.13.6.12

The developer can program all of the relations given above or only some of them. If all the relations have been programmed and parameterized, the user can still select any of their subsets for execution using parameters. The solution process is shown in Figure 6.13.6.1. The decomposition like this is obviously useful in any parallel processing.

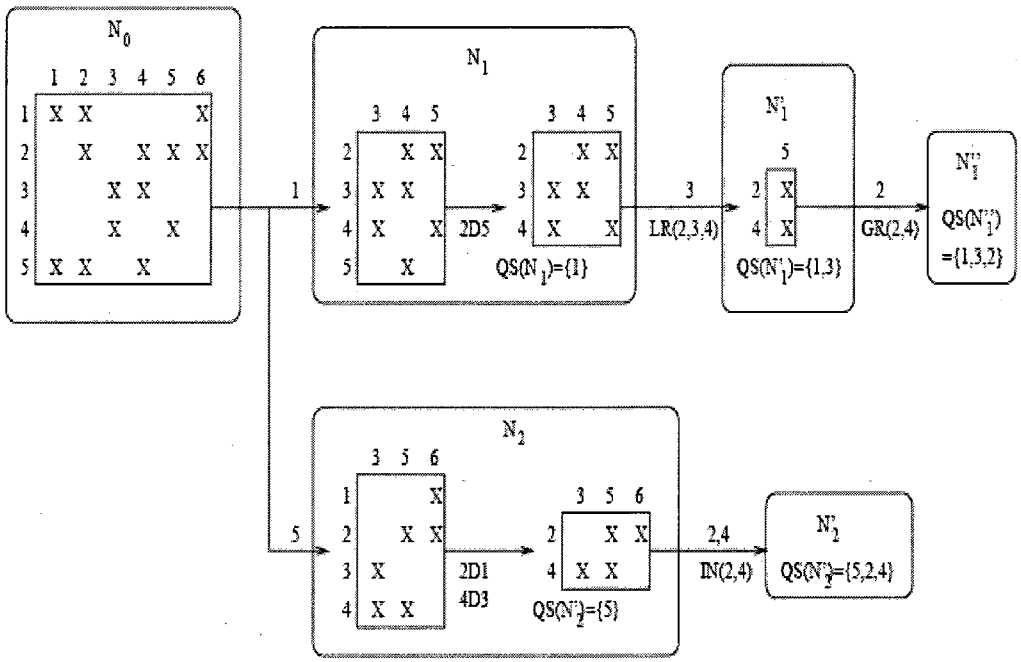


Figure 6.13.6.1: Final Search Method for the Table from Figure 6.13.1.

Column 1 and rows 1 and 5 are selected at the beginning ($GS(N_0) = \{1, 5\}$). After the selection of row 1 to $QS(N_1)$, row 5 becomes dominated by 2 (or 3) and is removed. The domination of descriptor 5 by descriptor 2 is denoted in the Figure 6.13.6.1 by 2D5. Now descriptors 2, 3, 4 are locally equivalent, denoted as $LR(2, 3, 4)$. One of them, say 3, is selected. Descriptors 2 and 4 then become globally equivalent in node N'_1, denoted as $GR(2,4)$. One of them, say 2, is selected. This leads to the solution $QS(N''_1) = \{1, 3, 2\}$.Now the backtracking to the initial node, N_0, occurs and descriptor 5 is selected. Next, descriptors 1 and 3 are removed since they are dominated and then descriptors 2 and 4 are selected as indispensable descriptors in node N'_2 that is denoted as $IN(2, 4)$ in Figure 6.13.6.1. This produces the solution $QS(N'_2) = \{5, 2, 4\}$. After backtracking to the initial node $GS(N_0) = \emptyset$, node N_0 is removed from the open-list. The open-list = $\emptyset$ completing the search of the tree. The last solution of the minimal cost 3 is then proven to be the optimal solution.

	1	2	3	4	5	6	7
A(4)			X	X			
B(3)			X		X		
C(3)							X
D(3)	X	X	X		X		
E(2)				X	X	X	
F(4)	X	X				X	

Figure 6.13.6.2: A Covering Table with Costs of Rows that are not Equal.

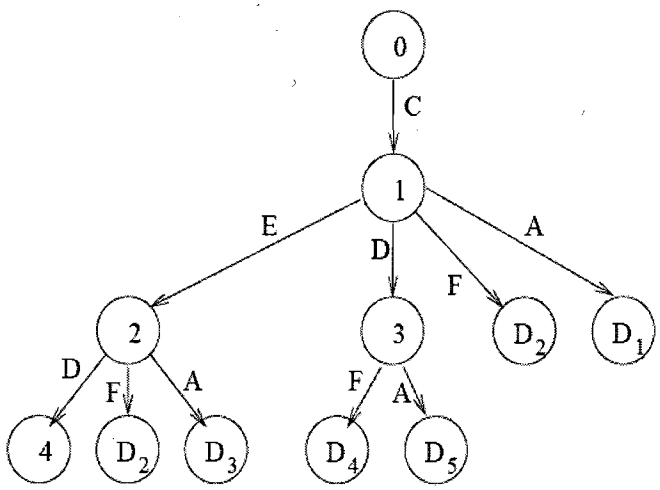


Figure 6.13.6.3: A Search Method for the Table from Figure 6.13.6.2.

6.13.7. General Ideas about Covering and Mapping Problems

In section 6.13.6 we showed few of many strategies for the unate covering problem. Similar approaches can be created to binate covering, SAT, even-odd covering and graph-coloring.

Note the following facts

- not all of the minimal solutions were obtained but more than one was produced,
- only node N_0 is permanently stored in the tree,
- if the user declared parameter $F_{min\ min} = 3$, the program would terminate after finding the solution {1, 3, 2}. In some problems, guessing or evaluating the cost of the function is not difficult.

If all of the above relations, except the most expensively tested local descriptor equivalence,

were declared, the complete tree consisting of 8 rows and 3 solutions would be obtained.

As illustrated in PPRM, FPRM, GRM and ESOP search examples from chapters 2, 3, 4, 6, 7, 8, 9 and 10 the same properties exist for other problems. In some problems very good results are found using Branch-and-Bound and Ordering as global strategies and *MUST0, EQUIV, REAPNT* to define the local strategy. *EQUIV* only checks for the global equivalence of descriptors. The covering table shown in Figure 6.13.6.2 will be solved in this example. The cost of each row is entered next to its respective descriptor. The costs of the rows are now not equal. The tree structured state-space for this problem is shown in Figure 6.13.6.3. The details concerning the node descriptions for this tree are also illustrated in Table from Figure 6.13.6.4. (By *pred(N)* we denote the predecessor node of node *N*).

The search starts from node 0 where no column is covered, so set *AS* consists of all the columns. All rows are available as descriptors. Initial *QS* is an empty set, since no descriptor has been yet applied. After being processed by *EQUIV*, it is found that descriptor *B* is dominated by the another descriptor *D*. Therefore, descriptor *B* is deleted from the descriptor list. *MUST0* finds that descriptor *C* is indispensable (with

respect to column 7), and it is then immediately applied by *GEN* to create the new node 1. The descriptor list is then ordered by *ORDER* using the quality function mentioned above. Assuming the coefficients $c_1 = 0.5$, $c_2 = 0$, and $c_3 = 1$, the costs of descriptors are

$$Q(A) = 2 + 4/2 = 4.0, \qquad Q(D) = 1.5 + 4/4 = 2.5, \qquad \textit{Equation 6.13.7.1}$$

$$Q(E) = 1 + 3/3 = 2.0, \qquad Q(F) = 2 + 4/3 = 3.3. \qquad \textit{Equation 6.13.7.2}$$

The descriptor list is arranged according to the descriptor costs as $\{E, D, F, A\}$. The descriptors are applied according to this sequence.

There is no difference between the application of the descriptors in the sequence of *E, D* or *D, E* for the solution in this problem. Therefore, if descriptors *E* and *D* have already been applied, it is not necessary to apply them again in another sequence. This is why descriptor *E* is cancelled for node 3; *E* and *D* for node D_0 ; as well as *E, D* and *F* for node D_1. This cancellation is done by *REAPNT* . The above procedure prevents node D_0 from finding the descriptor to cover column 5. Therefore, this node is not in the path to the solution and should be cut off by *GENER*. This phenomena also happen for nodes D_1, D_3, and D_5. The cost of node D_2, which is 13, exceeds the temporary cost *B* which is the cost of solution node 4 that has already been found. It was, therefore, cut off by the *Branch-and-Bound* strategy. So was node D_4. The whole

search procedure in this example deals with 11 nodes but only stores the descriptions of 5 nodes in the memory structure. A total of two optimal solutions were found in the search.

Observe that PPRM, FPRM, SOP, ESOP etc problems are **covering problems with various constraints.** They are all "subset selection problems", i.e. they are all formulated like that: "select such subset of a set of all sub-functions of certain kind that some constraints are satisfied and some cost is minimized". The SAT problem is also a subset selection problem, we have to select some subset of elements $\{x_1, \bar{x}_2, x_2, \bar{x}_2, ..., x_n, \bar{x}_n\}$ that a formula is satisfied SAT (x_1, ..., x_n) = **1**.

The **mapping problem** is to find such mapping X→ Y where X and Y are arbitrary sets that some constraints R_i (X, Y) are satisfied and some cost function on X and Y is minimized. Thus the subset selection problem in which Y = {0, 1} with meaning: 0 – not selected, 1 – selected is a special case of the mapping problem. This very powerful metaphor for problem solving helps to have a unified view to many practical CAD and AI/robotics problems that will be illustrated in next chapters with several examples, particularly for problems of interest to quantum CAD.

	0	1	2	3	4
N	0	1	2	3	4
SD	0	0	1	1	2
pred(NN)	-	0	1	1	2
OP	-	C	E	D	D
F	0	3	5	6	8
NAS	7	6	3	2	0
NQS	0	1	2	2	3
NGS	6	4	3	2	0
AS	1~7	1~6	1,2,3	4,6	-
QS	-	C	C,E	C,D	C,E,D
GS	A~F	E,D,F,A	D,F,A	F,A	-

Figure 6.13.7.1: Node Descriptions for the Tree from Figure 6.13.6.2.

The methodology presented in this chapter and illustrated with many examples in next chapters explains the characteristic trade-off relationship between the knowledge-based reasoning and the exclusively intrinsic search already mentioned in previous sections. The direct description of the problem allows us to find a solution based strictly on the generation of all possible cases that are not worse than the solutions generated previously (quantum or not). The successive addition of the information in the form of new heuristic directives and methodic directives that are based on the analysis of the problem and the solution process (e.g. quality functions, domination relations, equivalences, $F_{min\ min}$, etc.) allows for the search to be decreased. Adding a piece of information can decrease search dramatically, which especially important in quantum.

Until now, we have not focused on how the relation COV is *represented.* This could be an array, a list of pairs (r_i, c_j), a list of lists of columns covered by rows, a list of lists of rows covering the columns, various oracles, etc. The selection of the representation is independent from the selection of the method and from the strategy, but various combinations of these can have different effects. At some stage in creating the program or oracle, the user decides on the selection of, for example, the binary array and writes the corresponding functions. The user can then work on the representation of the array next: using words, or using bits. The arrays *M(N)* also do not necessarily need to be stored in nodes as separate data structures, they can be re-created from *AS(N)* and *GS(N)*. The local strategy parameters should also be matched to the representation. This is related to such factors as the total memory available for the program as well as the average times needed to select the node, to generate the node, to extend the node, to select the descriptor, and to check the solution condition.

Let us analyze one more example of a real-time system based on a parallel quantum computer.

6.14. Real-Time based Parallel Quantum Computer. A Hypothetical Scenario for QSPS

Assume we want to build a parallel quantum computer that calculates a trajectory for the US defense land-to-air missile from the received in real time data about the

approaching enemy missile. US counter missile should be fired in no more than 5 minutes to destroy the enemy missile. Otherwise it would be too late. If we use the optimizing Grover Algorithm on a single quantum computer the time of $O(\sqrt{N})$ may be longer than 5 minutes. So the calculated trajectory of US counter-missile result will be optimal but useless because it would be too late to destroy the enemy missile. Having however a parallel system with several Grover processors we can allow each of them to work with a different oracle and with a different number of Grover Loop iterations, making measurement in each processor after 10 seconds, 20 seconds, etc. Thus in the first 10 seconds we already have some trajectory solutions for the US missile, after 20 seconds we can get a better one for which to reprogram the counter missile, and so on. When the time to shoot comes after 5 minutes, we have already a solution selected among thousands of gradually improved solutions with more and more optimal trajectories. This is definitely practically better than to keep waiting for the forthcoming "optimal solution" while the enemy rocket is threatening to destroy US. In many situations like this an approximate solution available now is better than the exact solution obtained too late.

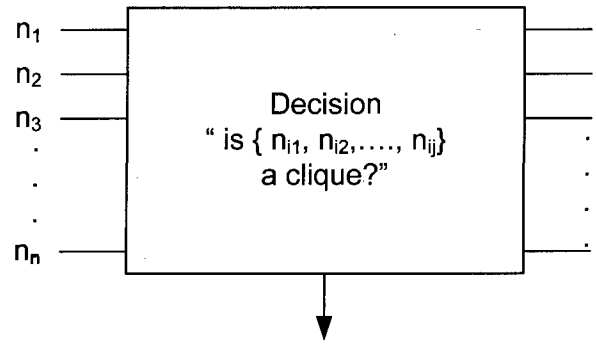
Several similar scenarios can be invented which demonstrate how to use the trade-off between the time of obtaining a solution on a parallel quantum computer and the quality of this solution. An optimal real-time system should take these trade-offs into account. This is a well-known problem from real-time control but it is applied here in a new way to a parallel quantum computer.

Let us also observe that in practice all problems are "real-time problems" when the computer technology has a flexible scale of providing solutions in time intervals from seconds to tens of years. Quantum technology is the only conceivable technology that will have this property. Let us give an example. Suppose that we want to factorize a big integer (related to the cracking of secret codes) using the Shor algorithm. If we use a standard computer working probabilistically the expectation of a correct guess would be some time longer than our Universe exists. So nobody would even try this approach. On the other hand, there exist integer factorization problems that a Shor Algorithm would solve in few minutes. Increasing these integers as problems given to Shor algorithm would increase the time of Shor algorithms solution times to hours, days, years and finally to the life-span of the human organization (like CIA) that requested this problem to be solved on the quantum computer. The probabilistic way of using Grover algorithm can find or not a solution in say 3 years when the optimal search would require 20 years. This situation may resemble catastrophic movies where some comet approaches the Earth and may crash so we need a supercomputer to find a necessary action to avoid the catastrophe. In case of the Grover Algorithm we would be thus, as a whole humanity, at the mercy of quantum measurements, which means at the mercy of probability. This is unfortunately a realistic situation similar to the metaphoric joke of the half-dead, half-alive cat of Schrödinger [Schrödinger26].

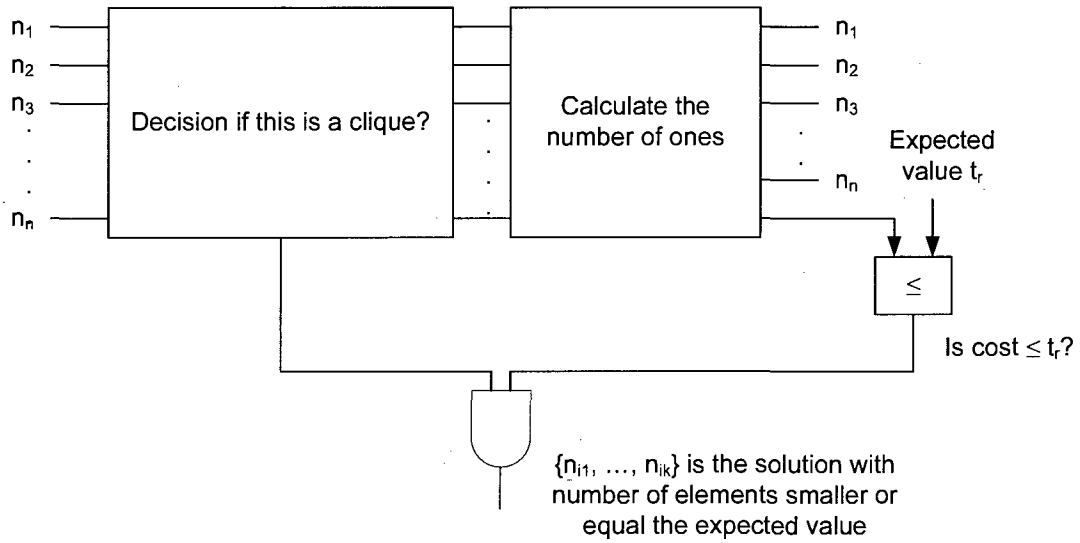
6.15. Variants of Quantum Computing in QSPS.

Standard Grover algorithm should iterate the Grover Loop of $\sqrt{N}$ number of times. There is however another possibility to use Grover, a probabilistic one. Let us take the graph coloring problem as an example. When the graph is very large, K nodes, and there is no any additional information about it, the number of colors should be assumed to be equal to the number of graph's nodes which gives $N = \log K \bullet K$ qubits for input variables. In such case $\sqrt{N}$ is a very big number and the Optimizing Oracle that uses the sorting/absorbing circuit (chapter 13) is both very complex and repeated very many times. In such case a better approach is to build a simpler graph coloring oracle composed only from the decision part – the Decision Oracle (Figure 6.15.1a). This oracle will generate randomly many solutions for each measurement. Running this simple oracle several times produces a solution with small cost (statistically). Next we can design the optimizing oracle with $N_1 = K \bullet n_1$ qubits where $n_1 << \log K$, thus reducing the time of running the Optimizing Oracle for Grover (Figure 6.15.1b).

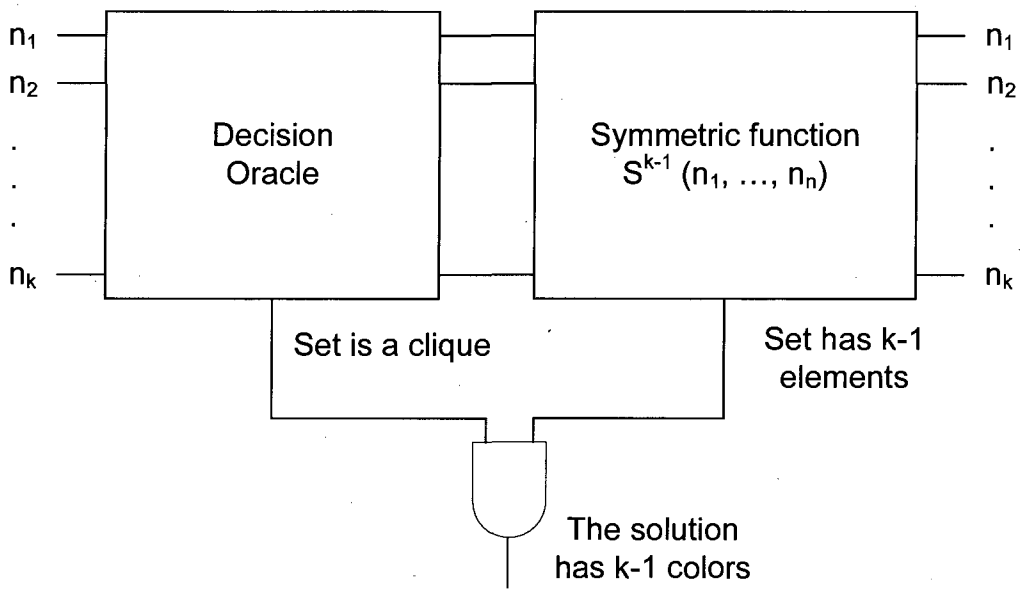
Finally instead of using the Optimizing Oracle one can build an oracle for predicted number of colors (we will call it the Predictor Oracle). Suppose that the Decision Oracle for a Maximum Clique Problem found a solution with k_1 colors. Then we can design a new oracle with the decision function as in Figure 6.15.1c.


n_1
n_2
n_3
n_n
Decision
" is { n_{i1}, n_{i2},...., n_{ij}}
a clique?"

(a)


n_1
n_2
n_3
n_n
Decision if this is a clique?
Calculate the
number of ones
n_1
n_2
n_3
n_n
Expected
value t_r
≤
Is cost ≤ t_r?
{n_{i1}, ..., n_{ik}} is the solution with
number of elements smaller or
equal the expected value

(b)



(c)

Figure 6.15.1: The oracles for the maximum clique problem. (a) the Decision Oracle , (b) the Optimizing Oracle, (c) the Cost-Predicting Oracle.

The given in this sections two examples of the sequence of oracles in Grover for "graph coloring" problem and "maximum clique" illustrate that the Grover algorithm that is normally used as a "decision maker" or "optimizer" can be also used as a "good guesser", at least for those problems that have many solutions. It can be also done for decision problems if the solution time is very critical, as in section 6.14.

There are many methods to combine the Decision Oracle, the Optimization Oracle and the Predictor Oracle with different number of Grover iterations. They can be all used in a general search system based on master-slave parallel processors, as the QSPSpresented in this chapter.

6.16. Heuristic Search versus Quantum Search

Above we discussed various aspects of search and its link to representation – how general are the search ideas, how related to quantum or non-quantum realization?

Observe that every CAD problem from our thesis has two aspects:

1. The concept of certain type of logic circuit type (such as structure, ancilla bits, types of gates, number of levels, etc) and the data structures in the synthesizing program that represent this circuit.

2. The method to search the space of solutions for the given specification of Boolean function and for the assumed type of the circuit.

Several concepts contributed to the search methods presented in this dissertation. The thesis is based on 20 years of experience of PSU group in optimizing AND/EXOR logic and reversible design as well as on recent papers from other groups. The thesis takes ideas from many previous papers and books: on one hand it expands on the optimization methods from [Dill01] and on the other hand on the quantum search paper [Li06] to build a uniform approach to quantum circuit synthesis based on search. This experience was reflected in the circuit types and search strategies for them.

The methodology of our previous software was applicable to traditional computer-automated digital design and synthesis, as well as for off-line Evolvable Hardware, including quantum hardware. The methodology from [Li06] is applicable to any problem described by an oracle. Our new methodology that was presented in this chapter is more general and incorporates the previous approaches as just its special cases.

The dissertation presents new search approaches:

1. The simplest is the classical Iterative Deepening Depth First search applied to logic synthesis of AND/EXOR reversible cascades based on affine gates. This is presented in chapter 7.

2. The second and more advanced is the ECPS search from chapter 6, more broadly applicable for general learning and solving combinatorial logic problems. It is used in chapter 8 for the minimization of GRM forms for incompletely specified Boolean functions.

3. The third is the quantum search QSPS (chapters 5, 6, 11 – 15), which is the main topic of this dissertation.

The Extended Cybernetic (Multi-Strategic Learning) Problem-Solving (ECPS) Algorithm was created based on my previous experiences with search algorithms. It expands on ideas from [Perkowski78, Perkowski82, Perkowski92, Perkowski99e, Perkowski02, Dill97, Dill97c] implemented in Multicomp and its next variants [Perkowski92, Software1, Software2]. Our new approach aids humans in designing application-specific solutions for binary and multi-valued logic synthesis and minimization problems. Most fundamentally, a more powerful state-space/evolutionary approach to solution derivation is employed in QSPS, for simplicity, generality and most importantly – to make a general link to quantum computing. When a problem is formulated as a search in some space, then it is next relatively easy to make variants of this search through evolutionary, quantum and probabilistic methods. The problem formulation, the cost function, constraints, heuristics, and other components of the specification are more important than the final representation of the search in one or another software, hardware or even type of computing (classical versus quantum, sequential versus parallel).

Our fundamental philosophy starts from the assumption that any combinatorial logic problem (or constraints satisfaction problem) can be solved by searching some space of known states (for instance, these states are the circuit structure instances being optimized). Solutions in this approach are achieved with an intelligent strategy using both human-designed heuristics and state-space search mechanisms [Nilsson98, Lugar02]. Our method includes evolutionary ideas but they are different from previous

Darwinian and Lamarckian learning approaches implemented in our group by Karen Dill, Martin Lukac, Normen Giesecke, Mozammel Khan and others. This is also in contrast to conventional evolutionary methods, that most often do not use the concepts of *"search in state space"*. One of our innovations is that of the two-level search which is based on the concept of polarity of spectral expansion of a Boolean Function. The upper level of the search performs the global exploration in the space of polarities, while the lower-level local search searches the best circuits for the given polarity. This lower level search can use any other method including evolutionary, A* search or simulated annealing. Therefore our approach from this chapter can be categorized as a memetic algorithm. The quantum search is presented in the general framework of sequential/parallel search as a sequence of exhaustive searches in reconfigurable systems with quantum Grover Algorithm based accelerators.

It is well-known that in the field of logic synthesis the researchers have several decades of experience producing useful human-designed software systems based on decision functions, butterflies and search, which we inherit to be used for the quantum CAD methodologies developed here. Thus, the search methods expertise must be combined with known quantum search algorithms, to make further progress, rather than to *"re-invent the wheel"*.

The previous experience of the PSU group with genetic methods [Dill97, Dill97a, Dill98, Perkowski99e] has shown that the evolutionary approach has both practical

solution time, quality of solution, and problem size limitations. For larger problems this approach creates only quasi-minimal solutions. It has just no means to achieve 100% convergence and the exact minimum of the cost function. Although the Genetic Algorithm (GA) and Genetic Program (GP) have the ability to adapt well to a particular function, they produce no explanation of design methodology and no rules of generalization for solving other problems. The GA/GP software does not learn a problem-solving strategy. Neither does it learn a general method for approaching a class of problems. For example, as the GA is applied to logic minimization [Dill97, Dill97a, Dill98] , after finding a good solution to one Boolean function, it approaches the next Boolean function to be minimized with no general learned knowledge. The same circumstance is also found in the application of the GP to logic synthesis [Dill97, Dill01]. On the other hand, the research on functional decomposition (Ashenhurst/Curtis decomposition and bi-decomposition) in the PSU [Files97, Files98, Files98a, Perkowski05], while creating good solutions, is not easily tunable to reversible and quantum technologies (at least we were not able to find a solution). Traditional exhaustive search mechanisms (breadth first, depth first, branch-and-bound, etc.) guarantee an optimal solution from the solution space, but are (often) prohibitively time consuming [Lukac04, Giesecke06]. Thus, both complete (searching the entire state-space) and incomplete (evolutionary and rule-based) search strategies may be unsatisfactory for producing a general problem solving technique for practical applications.

In contrast, the ECPS algorithm incorporates both pure and heuristic search strategies, and problem solving/learning paradigms, into a synergistic system. All learning methods are combined to form an intelligent, superset, "toolbox" of solution search space methodologies. This algorithm builds on the strengths of different search methodologies. First, within this new problem-solving algorithm, the problem-class-specific search strategies for logic minimization are built, for which the type and number of rules are selected. Then, within this training phase, a solution "pattern" (describing the search methods) is automatically designed for a problem class. This is done from analysis of the network, time available, stage of the design process, and limited user input. After the meta-algorithm has developed the solution pattern for a class of problems, any problem within this class may be applied. Finally, as the outputs are circuits, the combinatorial logic may also be depicted graphically as circuit, equation, truth table, K-map, or algebraic form, aiding the user in modifications of strategies, heuristic development, visualization of data, and the optimization process [Perkowski99e] (see also chapter 7). The visualization helps the intuition of the software developer.

For the purpose of comparison to other researches, the ECPS is applied to the GRM minimization problem in chapter 8. The results are given showing a comparison to those of Dill [Dill97a] and of Debnath and Sasao [Debnath95, Debnath96] for the minimization of completely specified GRM logic. Further, as the ECPS is capable of minimizing incompletely specified GRMs, its results are compared with that of Dill

and Perkowski [Dill01], the only other software designed for this purpose. Finally, the three search programs can be compared on other problems from this dissertation: in theory all three search approaches are applicable to all search-based methods from this thesis.

Concluding, here is the main philosophy related to search and developed in my dissertation:

1. A realistic quantum implementation technology has been selected and briefly presented (NMR) for which our circuits will be optimized on four levels: pulses, permutative gates, circuits and oracles (blocks, systems). The methods above the first level apply also to any type of reversible circuit realization (as illustrated in section 2.3 on cellular automata). Therefore, our circuit synthesis methods are very general and can be possibly applied for many new (reversible) technologies in addition to quantum. *To the circuits on all levels and in all technology variants we can use the same universal search methods, which are however tuned to each of the problems by the problem-specific cost functions*.

2. Powerful logic algebras have been generalized and invented to synthesize circuits optimized for the realistic cost functions. They combine the properties of linear independence, linear/affine decomposition, and the Reed-Muller logic hierarchy. Although these methods can be applied to non-reversible logic as well,

they are especially good for quantum realization since they are based on the assumption that NOT and CNOT gates are inexpensive with respect to multi-input Toffoli gates, which assumption was shown in Chapter 2 to be good for known quantum logic (NMR). (It may be not necessarily true for future quantum technologies, other reversible technologies (optical, CMOS adiabatic) and especially for standard VLSI where EXOR operator is not that cheap comparing to AND operator). *Uniformity of these concepts allows to create uniform search algorithms for all of them.*

3. The usefulness of these new invented by us approaches will be illustrated on several practical circuits realized for the selected NMR technology model. *The cost differences for some types of functions are already quite high on small examples that we tested.*

4. *A number of synthesis methods and techniques are invented and realized.* Not just one method.

5. Further, an analogy and extension of the entire sub-area of AND/EXOR logic is made to the Affine generalizations of the AND/EXOR circuits. Thus the Zhegalkin Hierarchy is extended.

6. *A new type of meta-algorithm for search on classical computer was ultimately invented.* This is referred to as the Enhanced Cybernetic (Multi-Strategic Learning) Problem-Solver (ECPS) Algorithm. Our software is compared to those of other authors.

7. The quantum oracles are shown for several combinatorial problems of CAD and used for synthesis of classical and quantum circuits. *This leads to a systematic general development methodology that uses Grover algorithm to accelerate CAD algorithms (chapters 12 – 14). Based on these ideas the QSPS quantum problem-solver was proposed and simulated.* Its practical power cannot be evaluated since quantum computers are now available for toy problems only. This approach can be also used for several constraint satisfaction problems in robotics (Chapter 15).

CHAPTER 7

Affine Binary Gates and Affine Circuit Structures

7.1. Introduction to the Concept of Affine Gates

In this chapter I will introduce the fundamental concept of this thesis – the affine gates. There are three basic types of such gates:

1. Affine Root of Not gates (ARNG) (chapter 7),
2. Affine Toffoli gates (chapter 7),
3. Affine Complex gates (chapter 9).

We can create big quantum gates more efficiently from these new primitives. These gates are next used in generalized cascades that include both Toffoli gates and new inexpensive interval quantum gates that are built from ARNGs.

Currently quantum cascades are built from CNOT gates (Feynman, 1-Controlled NOT) and n * n Toffoli gates (k-controlled NOTs, here $k \leq n - 1$). These realizations include very expensive gates when k is large [Maslov03]. Therefore we propose in this chapter some families of k-input gates that have inexpensive realizations in terms of the number of (truly quantum realizable) 2 * 2 gates. Each family has different interesting properties and should be used in conjunction with other families. For instance some of these families allow realizing every reversible single-output function of "even type" (with even number of true minterms) and should be used together with

standard k-controlled Toffoli gates to realize the so-called "odd type" functions (binary odd functions have odd number of true minterms).

It is well known that AND gate in classical standard logic is irreversible. Given the output, one can not obtain the definite input states. The input variables a and b can be 00, 01 or 10 and produce an output B of 0. Therefore a reproduction of the inputs is not feasible. The Feynman gate in Figure 7.1.1 preserves all information from the input to the output. Checking the truth table of this 2*2 gate, it can be observed that the input values can be constructed uniquely from the output values. This gate is inexpensive in all known to me quantum technologies and should therefore be a base of synthesis, which means, it should be used as often as possible by every reversible logic synthesis algorithm. The Feynman gate is linear because of its EXOR and is also affine as each linear function is affine.

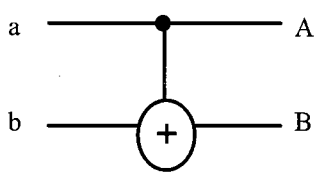


Figure 7.1.1: Feynman Gate; example for reversibility. This gate is a fundament of affine gates.

Besides the popular NMR quantum computers, Ion trap computers have become increasingly an attractive alternative [Nielsen00, DiVincenzo00]. Nature magazine

[Britton06] recently published an article where scientists (C. Monroe et al.) fabricated a micrometer-scale ion trap on a monolithic chip using semiconductor micro-electromechanical systems (MEMS) technology. They confined a single $^{111}Cd^{+}$ ion in an integrated radiofrequency trap etched from a doped gallium-arsenide hetero structure. If this steady progress marches on, then even skeptics will be convinced about this new way of executing quantum computation. A limitation on the number of qudits is not known yet, but is currently predicted to be much higher than in NMR [Nielsen00, DiVincenzo00]. Both NMR and ion trap allow realizing the so-called "Controlled Quantum Gates". The gate functionality is similar to that of a multiplexer. Additionally, it is not the input that the multiplexer selects (as there is only one input besides the select), but the function applied to this input. Concluding, all gates introduced below can be practically and inexpensively built in at least two quantum technologies, NMR and ion trap and in both these technologies CNOTs and CV/CV† gates were realized. Now we will present families of affine gates.

7. 2. Affine Root-of-NOT Gates (ARNG)

7. 2.1. Design of 3 * 3 gates and circuits using controlled gates.

Let us first look at the well-known Toffoli gate circuit from Fig. 7.2.1. It includes only 2 * 2 quantum realizable gate primitives. This decomposition is therefore close to real quantum hardware and allows good quantum cost approximations. Calculating the

number of 2 * 2 quantum gates as a pulse cost approximation is a good heuristic. Many circuits of this type were generated by Hung et al [Hung06], they use only 1-qubit gates – inverters and 2-qubit gates-controlled-V, Controlled-$V^{\dagger}$ and Controlled-NOT. Observing these circuits one can appreciate that all controls of V, $V^{\dagger}$ are linear or affine functions of variables or outputs of other macros. Affine binary function is a linear function or its negation. Analyzing these types of circuits and appreciating small relative cost of NOT and Feynman gates, we assume in this section that all controls are affine functions, which means, linear functions or their negations. It is easy to make "in-line" mirrors for affine gates.

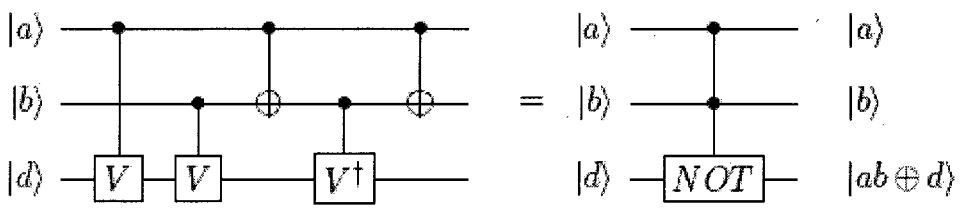


*Figure 7.2.1: The cost of a 3*3 Toffoli gate is five 2-qubit gates. On the right we see the symbol of Toffoli gate as a double-controlled NOT. Hence the another name of Toffoli gate as CCNOT gate.*

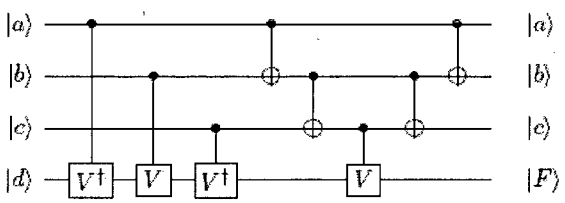


Figure 7.2.2: Realization of "double-cube" function $|F\rangle = (\bar{a}b\bar{c} + a\bar{b}c) \oplus d$

We do not care at this time how the upper part of the circuit, the control, is realized – we have developed elsewhere efficient methods for synthesis of such affine functions. The controlled (target) single qubit functions are inverters, V and V^+ gates in one variant and only V, V^+ in another variant. This way the 3-qubit Peres gate can be also created, as well as many other known gates. Peres is perhaps the least expensive universal binary permutative quantum gate (no proof exists yet, but nobody found a counterexample). This gate can be used instead of Toffoli in all our methods below.

As we see, the principle of our approach is simple. Knowing a powerful pattern of creating Peres and Toffoli gates, we use this pattern to systematically (or stochastically) generate new families of "interesting gates" under certain constraints of binary (permutative) realizability discussed below. Next these gates are used as macros in quantum circuits minimization. New gates are created by surrounding these macros with affine functions (CNOT, NOT). In the presented here variant of our CircuitSearch minimization program we use all affine functions as control functions and we use V, $V^\dagger$ (and NOT in some variants) in the data path (target) qubits. In case of 3 * 3 circuits it is relatively easy to use this approach to generate affine controls in variables a, b and c to generate the full Toffoli-like, Peres-like of "Fredkin-like" gates, in particular the gate from Figure 7.2.1. The question of course arises, "what is an interesting gate?" We will try to answer this question below, but let us observe first that interesting is a gate that reduces quantum costs when applied in synthesis of general or special types of Boolean functions. Gate patterns from Figure 7.2.1 and Figure 7.2.2 are "interesting". They create families of many useful affine gates by

inserting all possible combinations of V, V+ to target boxes. Let us now analyze the problem of synthesizing the 4 * 4 Toffoli-like (Toffoli family) gates and circuits.

7.2.2. Design of 4 * 4 gates and circuits using controlled root gates

CircuitSearch was created to aid development of "interesting" gates. Playing with our CircuitSearch program we create, for instance, the circuit from Figure 7.2.2 and find that it realizes the function $(\bar{a}\,b\,\bar{c} + a\,\bar{b}\,c) \oplus d$ which is a sum of minterms of Hamming distance 3 in three variables a, b, c exored with variable d. This is an interesting function with respect to the criteria mentioned above. We call it a dual-cube function. Using CircuitSearch in a smart way and critically analyzing the generated by it circuits and their truth tables we find more interesting functions that become the base of new circuit structures and our new synthesis algorithms for these structures. An interesting observation can be made by analyzing Figure 7.2.1. All component primitives (gates) used there are 2-qubit and the function realized on the lowest bit is ab $\oplus$ d. Each of controls can be multiplied by variable c to obtain solution abc $\oplus$ d. But now, gates V and V^+ need two controls. It means, that these gates should be rewritten again to 2*2 gates, but now the gates G = square-root-of (V) will be used instead of gates V and the gates square-root-of(V)-adjoint gates $G^\dagger$ will be used instead of gates $V^\dagger$ (Figure 7.2.3). Observe that this way we not only extend the Toffoli gate to 3 inputs in AND, but we create a general-purpose recursive method to generate Toffoli gates with any number of inputs, assuming availability of 2^k-root-of-V gates.

Each of controls in Figure 7.2.1 can be multiplied by variable c to obtain solution abc ⊕ d realized in Figure 7.2.3a. But now, gates V and V^+ need two controls. It means, that these gates should be rewritten to 2×2 gates, but the gates G = square-root-of (V) will be used instead of gates V and the gates square-root-of(V)-adjoint gates $G^\dagger$ will be used instead of gates $V^\dagger$ (recall the G and $G^\dagger$ gates from chapter 2). We extended the Toffoli gate to 3 inputs in AND, but we have a new problem, "how to design the controlled gate controlled by two inputs ?". But this problem is similar to the one we already solved in Figure 7.2.1. Therefore, we deal here with certain type of recursion that we want to use generally in synthesis. Observe also that the control of each multi-controlled gate is an affine function (in this case it is even linear).

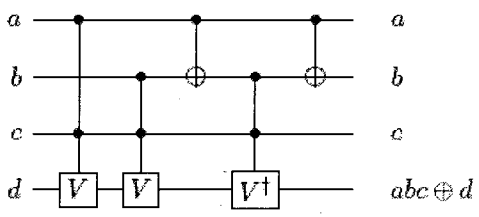


Figure 7.2.3: (a) Extension of standard Toffoli gate to 4×4 Toffoli gate by multiplying by signal c.

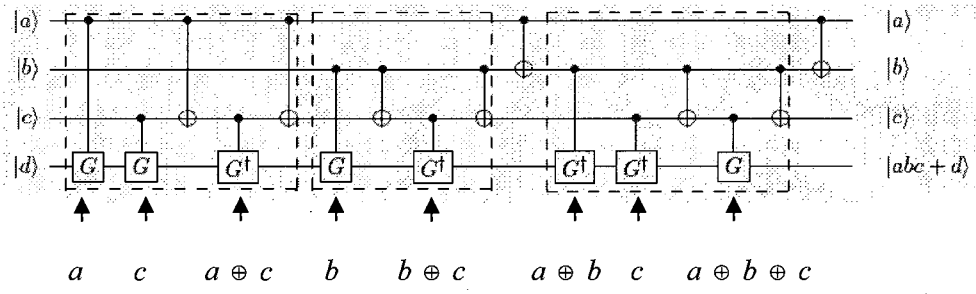


*Figure 7.2.3: (b) Realization of the 4*4 Toffoli gate from Figure 7.2.3a using controlled-root-of-order-four-of-NOT gates, CG. Linear controls are written for all $G/G^\dagger$ gates under them to simplify the analysis. The blocks shown with interrupted lines*

show the initial gates drawn according to the design from Figure 7.2.1 with additional multiplication by c (from Figure 7.2.3a).

To realize $abc \oplus d$ *we* have to realize each double-controlled V gate using 2*2 gates. This is done as in Figure 7.2.3b, each gate G represents square-root-of-V and thus the fourth-order-root-of-NOT. Similarly, the controlled hermitian gates CCV are built in Figure 7.2.3b using CG and $CG^{\dagger}$ gates. The circuit from Figure 7.2.3b using quantum simplification rules can be transformed to a simpler circuit from Figure 7. 2.4. This way our method re-invented the CCCNOT circuit found by Barenco (the triple-controlled NOT gate).

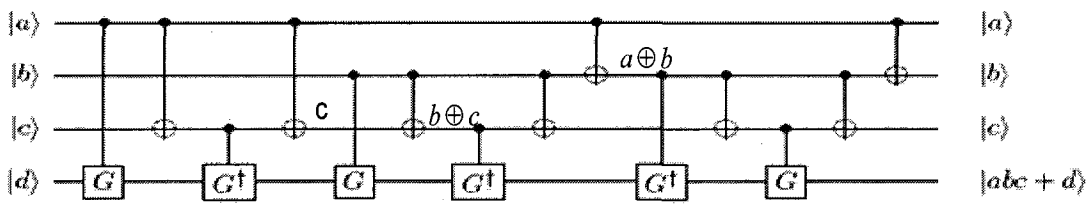


Figure 7.2.4: Simplified circuit from Figure 7.2.3b. Rule $G.G^{\dagger} = I$ was used for gates G, $G^{\dagger}$ controlled by c in Fig. 7.2.3b. Two gates from Fig. 7.2.3b have been thus reduced. Observe that this circuit has only 6 controlled $G/G^{\dagger}$ gates, each controlled by a linear function. This is one more example of Affine Root of NOT gate.

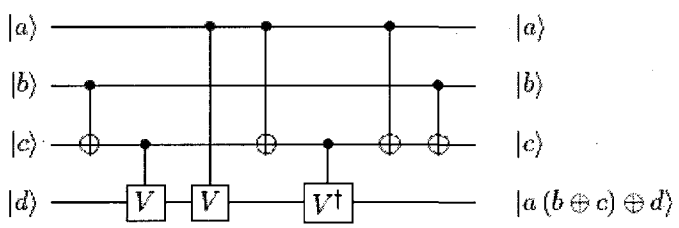


Figure 7.2.5: Realization of function $a(b \oplus c) \oplus d$ using linear controls of $V/V^{\dagger}$ gates.

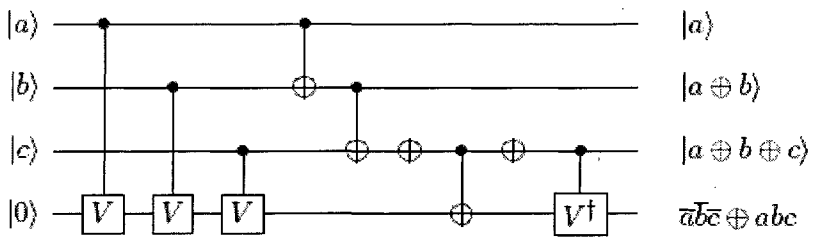


Figure 7.2.6: Realization of function $f = \bar{a}\,\bar{b}\,\bar{c} \oplus a\,b\,c$ *using affine-controlled target gates V,* $V^{\dagger}$ *and NOT.*

Figure 7.2.5 shows example of a set of linearly-controlled V/$V^{\dagger}$ gates which together realize the factorized Positive Polarity Reed-Muller (PPRM) form functionally equivalent to a sequence of Toffoli gates. As we see, we do not need to find the PPRM and next factorize it to find this circuit. We can just control gates V and V+ using linear (in general, affine) gates and next restore the original input values by the use of mirror gates. This method can be generalized to use arbitrary affine controlled gates and arbitrary mirror circuits.

Figure 7.2.6 realizes a double cube function. This is a pair of Hamming-distance-3 minterms on variables a, b, c but the minterms are different than in Figure 7.2.2 because of using other affine functions directly controlling the output target qubit d=0. Figure 7.2.7 presents the realization of function $(ab + ac + bc) \oplus d = ab \oplus ac \oplus bc \oplus d$. Many variants of the CircuitSearch program can be created for various types of controlled gates, controlled gates and realizability constraints. The control functions may be for instance all products of literals like $\bar{a}\,\bar{b}\,c$, or all functions of 3 variables. Similar

circuits can be build using controlled-square-root-of-NOT, controlled-fourth-order-root-of-NOT and in general controlled 2^k-root-of-NOT for k = 2, 3, 4, 5...

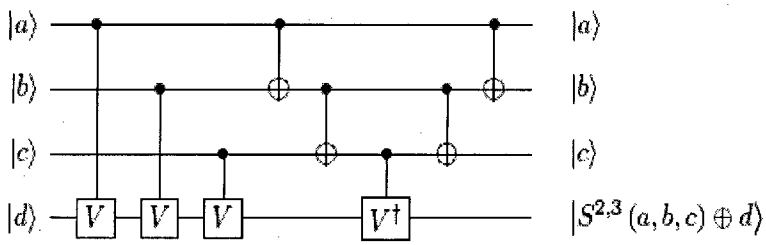


Figure 7.2.7: With d=0 we realized here a symmetric function of variables a, b, c. Observe that + can be replaced with ⊕ in the formula for S^{23} (a, b, c). maj (a,b,c) = S^{23} (a, b, c) = S^2 (a, b, c) + S^3 (a, b, c) is a totally symmetric function of a, b, c.

7.2.3. Design of big gates using Controlled-root-of-NOT gates

By big gates we will understand gates with 5 or more qubits. The costs of such gates increase, sometimes even exponentially, so their efficient design is very important. Such gates are very expensive in quantum realization so we will try to find inexpensive big gates and use them as much as possible as macros in synthesis. For instance, the 5 * 5 Toffoli gates are very expensive as quantum circuits since the realization of AND with many inputs requires many auxiliary gates and their mirror gates. We will illustrate this fact below. An arbitrary 3-controlled operator U can be realized using two 2-controlled Toffoli gates and a 2-controlled U gate as in Figure 7.2.8. Next each of the 2-controlled Toffoli gates is replaced as in Figure 7.2.1 and the 2-controlled U gate is realized similarly as in Figure 7.2.1, leading to the circuit from Figure 7.2.9.

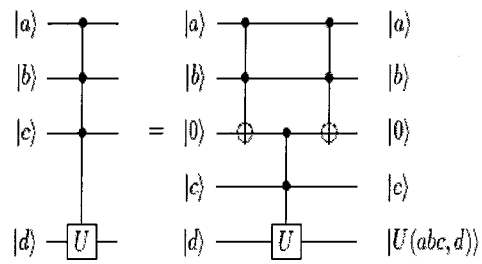


Figure 7.2.8: Realization of 3-controlled U.

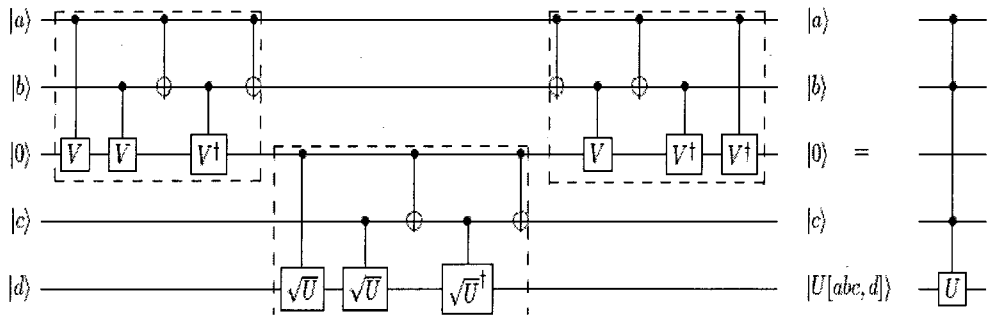


Figure 7.2.9: Realization of 3-controlled operator U from Fig. 7.2.8 with CV, CV$^+$ and Controlled $\sqrt{U}$, $\sqrt{U}^+$ gates. Pay attention to the mirror circuit top right.

Concluding, the realization of the 3-controlled U using quantum-realizable primitives in the space of 5 qubits is shown in Figure 7.2.9. Assuming U=NOT, the single product of 3 literals costs 15 2×2 gates while on the other hand two such products in Figure 7.2.2 cost only 8 2×2 gates. The method illustrated in Figure 7.2.8 and Figure 7.2.9 allows to design recursively any Toffoli-like multi-input gate building a structure from quantum-realizable 2*2 primitives. This way, any quantum circuit built in PPRM, FPRM, GRM or ESOP styles using Toffoli gates, CNOT gates and inverters is converted to a quantum realizable quantum array. But this method may create unnecessarily expensive circuits. Thus we will concentrate now on cheaper

realizations of gates for quantum cascades. The methods given in sections 7.2.2 and 7.2.3 are however still necessary for odd functions, such as a single minterm in the full space of products of literals.

7. 2.4. Design of 2-interval gates

An important subgroup of ARNGs are the 2-interval gates introduced for the first time in this section. Barenco et al [Barenco95] in their paper (which is one of the most cited papers in quantum literature) introduced the method to build 3 * 3 Toffoli gates using controlled $V/V^{\dagger}$ and 4 * 4 Toffoli gates using controlled $G/G^{\dagger}$ gates. They verified the solutions but they did not present a general approach to build arbitrary functions of this type. Also they did not discuss how to design those big functions that are especially inexpensive. We achieve these two tasks in this thesis.

1	1	2							$S^{2}(a,b)$	Barenco's Toffoli gate
2	1	2	3						$S^{2,3}(a,b,c)$	New gate
3	1	2	3	4					$S^{2,3}(a,b,c,d)$	New gate
4	1	2	3	4	5				$S^{2,3}(a,b,c,d,e)$	New gate
5	1	2	3	4	5	6			$S^{2,3,6}(a,b,c,d,e,f)$	New gate
6	1	2	3	4	5	6	7		$S^{2,3,6,7}(a,b,c,d,e,f,g)$	New gate
7	1	2	3	4	5	6	7	8	$S^{2,3,6,7}(a,b,c,d,e,f,g,h)$	New gate

Table 7.2.1: The schematic explaining construction of 2-interval functions of positive literals. Observe that all these functions are symmetrical. The table can be continued for any number of qubits.

This section has the main inspiration from the basic Barenco Smolin circuit from Figure 7.2.1. We started from this circuit but we also generalized our ideas to create a

theory for synthesizing arbitrary multi-input, multi-output functions using controlled root gates. As the first generalization, we extended, for more inputs, the Barenco circuit keeping the same structure of the circuit. Here in Table 7.2.1 we list the first seven of these circuits which we will call from now on the "2-interval circuits", as their structure is that of symmetric interval functions with two indices present and next two indices absent, and so on, as shown in Table 7.2.1. Unfortunately not all symmetric functions can be realized that way, so we will have to add more components to our cascades to create larger families of component functions to realize arbitrary functions. This concept is new not only in the realm of quantum circuit design but it is in general a new logic synthesis concept. Observe please, that the circuit from Figure 7.2.7 does the same to four qubits as the Barenco circuit from Figure 7.2.1 does to 3 qubits. Both these circuits have the same pattern. The first circuit realizes S^2 (a,b) $\oplus$ d in its lowest bit, while the second circuit realizes $S^{2,3}$(a,b,c) $\oplus$ d. Our program generates all the functions from Table 7.2.1 as truth tables, among many others. Patterns of 2-interval and double-cube gates can be proved for an arbitrary number of inputs. Amazingly, the 2-interval functions are exactly the same as the so-called "eigenvalue functions" introduced independently by T. Sasao [Sasao07]. In addition, from each function from Table 7.2.1 we create a family of functions represented as gate macros by inserting symbols V, V+ in all possible ways to target boxes (represented by small rectangles in Figures). An interesting example of inexpensive function of five variables is presented in Figure 7.2.10. Observe that all controls are affine and all controls are restored to input variables by using mirror

CNOT gates. The whole function from this Figure is a permutative function that can be used as a component (subfunction) of an arbitrary function realized by a quantum cascade.

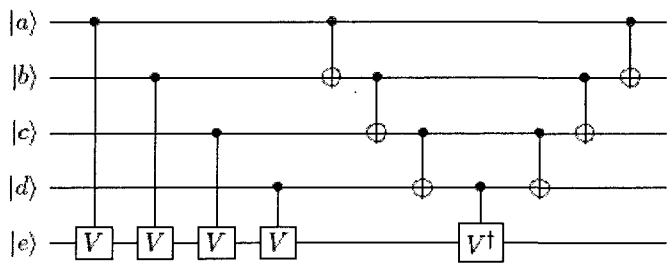


Figure 7.2.10: Realization of $S^{2,3}(a, b, c, d) \oplus e$ using ARNGs. Observe the same general pattern of connections as in Figs. 7.2.1 and 7.2.7.

If we would realize functions from Table 7.2.1 using standard multi-output Toffoli gates and next macro-generate them to 2*2 quantum primitives as in section 7.2.3 the costs would be very high. The 2-interval and similar functions (and their derivative families) we call "cheap functions" because we use only CNOTs, CVs and $CV^{\dagger}$s in them, and we achieve these designs only by controlling single gates. Whenever we have to control with non-affine controls, it becomes more expensive, we have to add mirrors, sometimes ancillas and so on.

A question may arise, given an arbitrary function, how can I use my inexpensive special circuits to realize some functions to be used as components of arbitrary functions. The following theorem is of help.

Theorem 7.2.1:

Assume that:

1. A binary n-input, m-output Boolean function F is to be realized in a quantum cascade.
2. We assume the width n+m of the cascade. The cascade has n input qubits (that can be factorized and reuse) and m output qubits (that can be factorized and reused) and no more intermediate qubits.
3. We assume that mirror circuits can be used, multiple times if necessary, for every qubit to restore its value to the input value or some intermediate value.
4. A finite set of 2^n binary base (Linearly Independent, orthogonal) functions on n variables are given.

Then function F can be realized in a quantum cascade using only Toffoli, Feynman and NOT gates where each output of F is realized as an EXOR of subfunctions selected from the base functions and a constructive method of selecting these functions exists.

Proof. It has been proved by Marek Perkowski [Perkowski95] that for every $2^n * 2^n$ orthogonal binary matrix M representing a set of 2^n binary base functions there exists exactly one expansion of arbitrary n-argument Boolean function F in this base where the coefficients of the inverse matrix M^{-1} give the values of spectral coefficients of these base functions. The operations of multiplying rows by columns of such matrices and multiplying rows by column vectors are number-by-number EXORs (Modulo 2

additions). Thus for every set of orthogonal (Linearly Independent) functions we can find one unique solution in the form of an EXOR of those base functions that the spectral coefficients are equal 1 (look for examples in chapter 9). However, if base functions are arbitrary, then:

1. They can be very expensive to realize
2. They may require more ancilla bits than n+m.

Therefore we restrict ourselves only to those base functions that:

1. Are inexpensive as built from affine gates of this chapter
2. Allow to be realized without more than n+m ancilla bits.

Of course, the theorem and based on it synthesis method can be extended to all base functions of all families but this would lead to many ancilla bits and also the number of families of base functions is extremely large so it is more reasonable to restrict our method only to some families. Thus considering only inexpensive families is a good idea.

Affine gates are very useful to create gates for base functions to be used in new extensions of MMD algorithm [Miller03] or any other algorithm for quantum array synthesis, because our method creates affine gates for any number of inputs. Observe that in circuits minimized using standard ESOP (Exclusive Or Sum of Products circuits) minimization techniques only the Toffoli-like gates are used, i.e. k*k Toffoli, CNOT and NOT. But in the proposed method there are many more base functions

which in addition have small quantum costs. For instance, based on the sections covered so far we can use as base functions all the new double-cube, 2-interval gates and other cheap gates built from macros. With next sections and chapters we will add more inexpensive base functions to be used as new base families. This idea is new and specific to quantum circuits, because cost functions based on 2*2 primitives did not exist in classical and reversible logic.

7.2.5. Affine Toffoli Gates.

The second class of the (binary) affine gates that we invented are the Affine Toffoli Gates (ATG). Example of such a gate is shown in Figure 7.2.11. As we see, the Toffoli gate is surrounded with Feynman gates in such a way that the original argument variables a, b, c, d are restored on the outputs of the entire affine Toffoli gate. Thus these input variables can be reused directly be the next gates in the cascade. The Feynman gates on the left serve to create local linear preprocessors and the Feynman gates on the right are mirror circuits to restore the original argument values. This construction method is very general. The same types of gates are used in Polarity-Based Affine Forms. The gate from Figure 7.2.11 can be for instance treated as a special case of such a form with the first column as the affine preprocessor and the last column as the mirror affine postprocessor. This ATG gate is a very powerful generalization of a Toffoli gate for any number of inputs. It should replace Toffoli gates in all future synthesis algorithms. Observe that Fredkin and Miller gates are also

special cases of ATGs as they are created by surrounding Toffoli gate with Feynman gates.

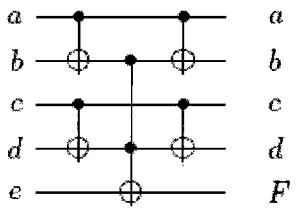


Figure 7.2.11: Binary Affine Toffoli Gate for function from Figure 7. 2.12.

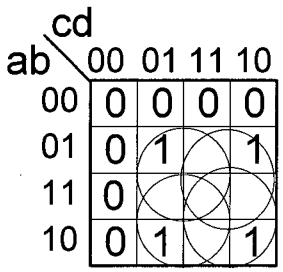


Figure 7.2.12: Graphical Analysis of the affine Toffoli gate from Figure 7.2.11.

As we see in Figure 7.2.12, four minterms of four variables each are realized in just a single gate with quantum cost 5 of 2*2 gates (for the Toffoli gate) plus 4 (for four Feynman gates). The total cost 9 is very small when compared to the cost of 4 Toffoli gates to realize minterms separately (which are 2-input, 3-input or 4-input, depending on quality of AND/EXOR minimizer applied). Such AND/EXOR circuit uses product groups that are created by flattening of the formula originating for F directly from Figure 7.2.11. The KMap from Figure 7.2.12 shows a characteristic pattern of true minterms for this kind affine Toffoli gates. Our synthesis software finds such patterns, but they can be also found from KMaps in "hand synthesis method". Figure 7.2.12 can

be explained algebraically as in Figure 7.2.13. As we see the Affine Toffoli gate is the cheapest of all realizations of the KMap from Figure 7.2.12.

$$(a \oplus b)(c \oplus d) = ac \oplus ad \oplus bc \oplus bd \quad (2-controlled \quad Toffoli)$$
$$= \overline{a}\, bd \oplus \overline{a}\, bc \oplus a\overline{b}\, d \oplus a\overline{b}\, c \quad (3-controlled \quad Toffoli)$$
$$= \overline{a}\, b\overline{c}\, d \oplus \overline{a}\, bc\overline{d} \oplus a\overline{b}\, \overline{c}\, d \oplus a\overline{b}\, c\overline{d} \quad (4-controlled \quad Toffoli)$$

Figure 7.2.13: Derivation of various non-optimal circuits for the minimum gate from Figure 7.2.11.

Concluding this chapter so far: we can create many types of inexpensive gates to be used in quantum cascades, they are all based on the concept of affine gates which are used in various ways to control other gates, such as classical permutative gates and truly quantum gates such as V.

7.3. More on Affine Gates

Now that we explained briefly the main ideas of our approach, more details will be given necessary to understand our methods used in this and next chapters.

7.3.1. Design of 3 * 3 gates and circuits using controlled gates.

Let us first look again to the well-known Tofoli gate circuit from Figure 7.3.1.1. It includes only 2 * 2 quantum realizable gates. It is close to real quantum hardware. So calculating the number of 2 * 2 quantum gates (primitives) as the "quantum pulse

cost" approximation is a good heuristic. We will use this heuristic in many examples that follow. Many circuits of this type were generated by Hung et al [Hung06], they use only 1-qubit gates – inverters and 2-qubit gates – controlled-V, Controlled-V^+ and Controlled-NOT. Observing these circuits one can appreciate that all controls are linear or affine functions. Although the method given by Hung et al gives exact minimum solution, it is very time consuming and thus restricted to small circuits. The methods that will be presented in this thesis can solve all examples from [Hung06] with much less effort and find approximate solutions for big functions quickly. These new methods are however all not exact, they do not give guarantee of the minimal cost. However, in all small examples that we considered (including those from [Hung06] the costs were very close to minimal.

Analyzing these types of circuits and appreciating small relative costs of NOT and Feynman gates, we assume below that all controls in our approach will be affine functions – linear functions and their negations. Let us observe in Figure 7.3.1.1 that the last CNOT on the right has no effect on the output in qubit c. It serves the only role of restoring the input b to its original state. This is not always necessary (as shown in Figure 7.3.1.3). Figure 7.3.1.2 illustrates two points of view on a macro. Its internal view with 2*2 quantum primitives and its external view as a permutative gate. Remember that in all next examples we will use these views and we should macrogenerate larger gates to the level of 2×2 primitives. To save the time and space we will not do this in most figures, however doing this would allow the reader to

appreciate the real gain in terms of quantum costs of the circuits designed by us in this and next chapters.

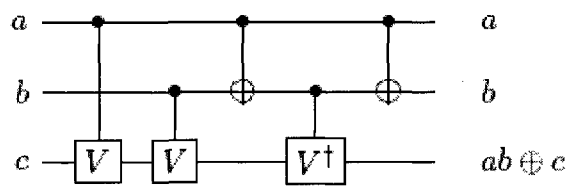


Figure 7.3.1.1: Realization of Toffoli gate with output logic equations. Only 2×2 gates are used.

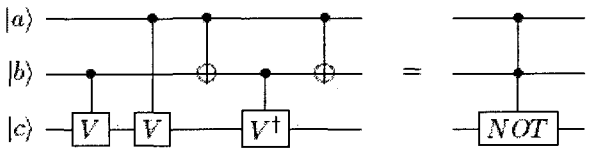


Figure 7.3.1.2: The cost of Toffoli gate is five 2-qubit gates. On the right we see the symbol of Toffoli gate as a double-controlled NOT. Hence the another name of Toffoli gate as CCNOT gate.

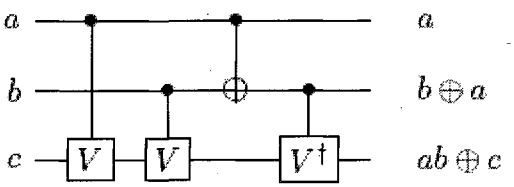


Figure 7.3.1.3: Peres gate has a cost of four 2-qubit gates.

7.3.2. Design of 4 * 4 gates and circuits using controlled root gates

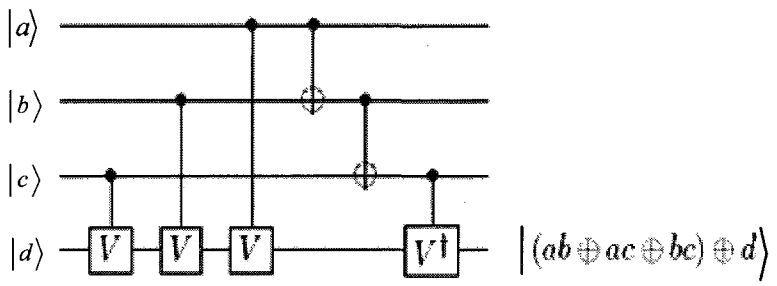


Figure 7.3.2.1: With d=0 we realized here a symmetric function of variables a, b, c. Observe that + can be replaced with ⊕ in the formula for S^{23} (a, b, c). S^{23} (a, b, c) = S^2 (a, b, c) + S^3 (a, b, c) is a totally symmetric function of a, b, c.

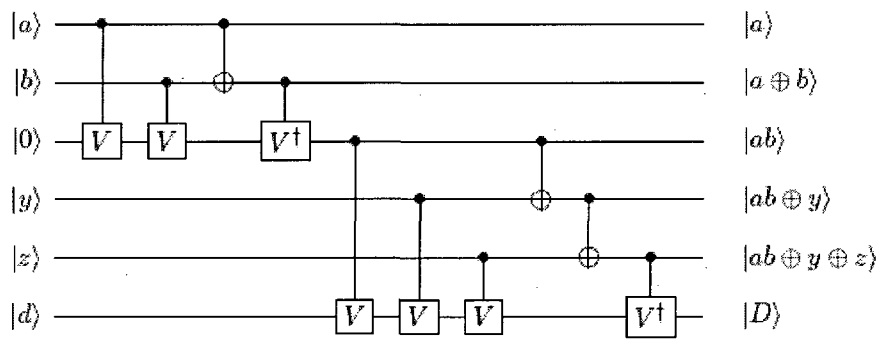


Figure 7.3.2.2: Realization of function D = maj (x,y,z) ⊕ d = [(ab)y + (ab)z +yz] ⊕ d. Please note the role of the ancilla bit |0⟩ in the third qubit from top. This entire circuit requires just one ancilla bit.

Figures 7.3.2.1 and 7.3.2.2 shows arrays with more than 3qubits build efficiently with affine-root-of-NOT gates. Similar circuits can be build using controlled-square-root-of-NOT, controlled-fourth-order-root-of-NOT and in general controlled 2^k-root-of-NOT for k = 2, 3, 4, 5….

It is well known that an arbitrary two-controlled operator U can be realized as shown in Figure 7.3.2.3. Here the operator $U = W^2$, where W is an arbitrary unitary operator. This circuit assumes that $W * W^+ = I$ and $W * W = U$. The circuits like in Figure 7.3.2.3 is a general prototype that can be further generalized in two ways:

1) to binary permutative circuits with more than two control wires (this chapter),

2) to multiple-valued permutative circuits, such as multiple-valued Toffoli, SWAP or Feynman gates (chapter 10).

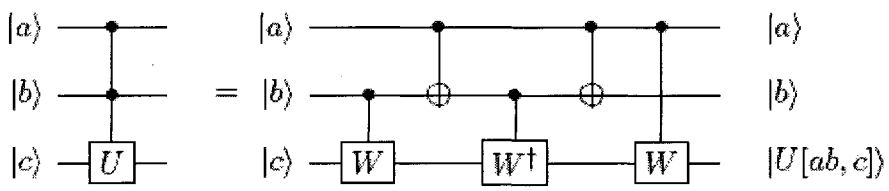


Figure 7.3.2.3: Realization of 3-input double-controlled U gate with use of two-qubit gates.

Let us first observe that all existing synthesis/optimization methods for quantum and reversible (permutative quantum) circuit synthesis (Cosine-Sine decomposition, De Vos, Miller and MMD, Perkowski et all) use Toffoli gates with more than 2 controls. These gates are often counted as having the cost of one, but in reality they are very expensive when realized with 2-qubit gates and we know that only 2-qubit gates are truly quantum realizable. Gates with many controls can be recursively decomposed as shown in Figure 7.3.2.4. In this figure the 4–controlled U is replaced with two 3-controlled NOT gates and a single 2-controlled U. The 2-controlled U can be next

realized as shown in Figure 7.3.1.2 and the 3-controlled NOT gates can be decomposed again as in Figure 7.3.2.4. This solution requires adding one ancilla bit.

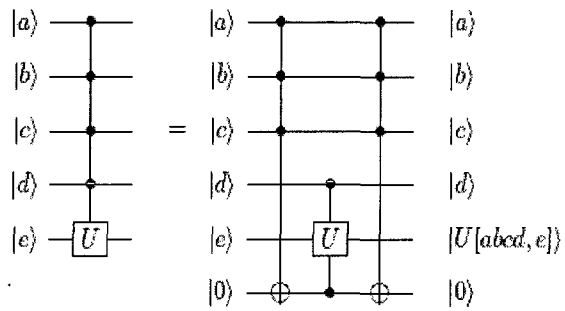


Figure 7.3.2.4: Realization of n-controlled U with 2-controlled U and two (n-1) controlled inverters. This approach requires only one ancilla bit.

Theorem 7.3.3.1

Every single-output Boolean function of n variables can be realized with n + 1 bits (One ancilla bit) using only 2×2 quantum gates.

Proof.

Every function of 2 variables can be realized in 3 qubits as an ESOP or similar form using 3*3 Toffoli gates with 2 controls. Next each Toffoli gate can be transformed to a combination of 2*2 primitives as in Figure 7.3.1.1. Similarly any function of 3 variables can be realized as an ESOP using 4*4 Toffoli gates, each realized as in Figure 7.2.3. When function has more than 3 variables it can be recursively macro-generated to smaller blocks using methods from Figures 7.2.8, 7.2.9, 7.3.2.3 and 7.3.2.4.

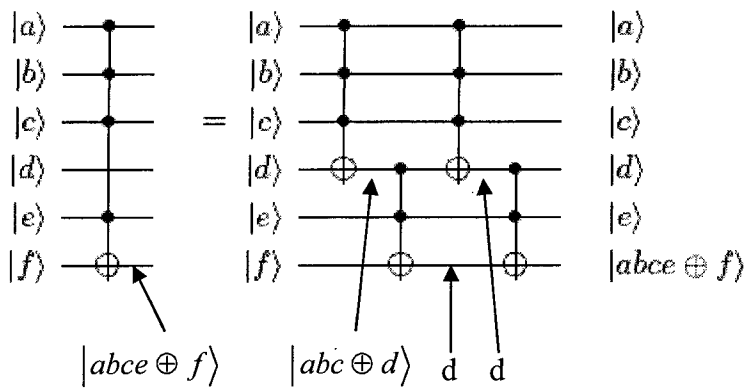


*Figure 7.3.2.5: Realization of (n-1) controlled NOT for a (n + 1) * (n + 1) width of quantum register. Pay attention to smart use of two mirrors.*

To illustrate this theorem, for instance, the 4-controlled Toffoli in the space of 6 qubits can be realized as shown in Figure 7.3.2.4. Two 3-controlled and two 2-controlled Toffoli are introduced. Next each of the 3-controlled Toffoli gates is replaced with a structure of 2-controlled Toffoli gates. Finally all 2-controlled gates are converted as in Figure 7.3.2.3 to quantum-realizable 2 * 2 primitives.

Similarly, arbitrary 3-controlled operator U can be realized using two 2-controlled Toffoli gates and a 2-controlled U gate as in Figure 7.2.8. Next each of the 2-controlled Toffoli gates is replaced as in Figure 7.3.1.2 and the 2-controlled U gate is realized as in Figure 7.3.2.3.

The methods illustrated in section 7.3 allow to design recursively any Toffoli-like multi-input gate using a structure of quantum-realizable 2×2 primitives. This way, any of PPRM, FPRM, GRM, ESOP or factorized ESOP circuits could be converted to a

quantum realizable quantum array. However, as shown in this, previous and next sections of this chapter, the designs of many functions can be improved.

7.4. Design of symmetric functions

Designing symmetric functions is useful in many practical problems. Symmetric functions are also easier to analyze than arbitrary functions. Therefore we analyzed design of symmetric functions using our methods.

We will use various definitions of symmetry of Boolean functions.

Definition 7.4.1: A Partially Symmetric function with respect to variables a and b is a function f(a, … b, ….) that if you replace in the formula a with b, you get the same function. If a function is symmetric with respect to every possible pair of its input variables then this function is called totally symmetric.

This is the simplest classification of symmetric functions which definition we use in this chapter. But there are many definitions of more powerful symmetries in functions that we do not use yet. For instance when any subset of variables can be negated or not, we have polarity concept which has 2^n symmetric polarities. Then we have the generalized Lattice Symmetries [Perkowski97]. We create exors of two cofactors, exors of three cofactors, exors of four cofactors, etc and compare them for equality.

We are comparing exors of cofactors in all possible ways: this is the most general classification of symmetric functions. All these symmetry based methods are basically related to restricting search. If we have a symmetric function, or a unate function or some special function type then it becomes possible to use mathematics to somehow restrict the search to find the minimal circuit for this function. We want to minimize these types of functions and we want to minimize the numbers of ancilla bits for various choices of quantum costs. That means we want to do everything possible to avoid using standard large Toffoli gates: the more inputs, the more expensive these gates are. These ideas can be useful to create gates, gate libraries and circuits. Below we use only some subset of these ideas.

7.4.1. Methods to analyze totally symmetric functions.

The interval functions from section 7.2.4 are all symmetric. Let us think what is the function $S^{2,3}$ of (a,b,c)? Let's show for three control variables a, b, c. First, we will generalize this pattern, we take every argument input variable to control separate V gates and we create an EXOR of all these controls to control $V^{\dagger}$. We can reconstruct the original signals in input variables but we do not care about this issue in general when we discuss single-output gates. We care only about the data path qubit: how it is controlled. We can analyze this circuit to learn more (Figure 7.4.1). We can generalize this pattern from Figure 7.2.10 to Figure 7.4.1 and next to Figure 7.4.2.

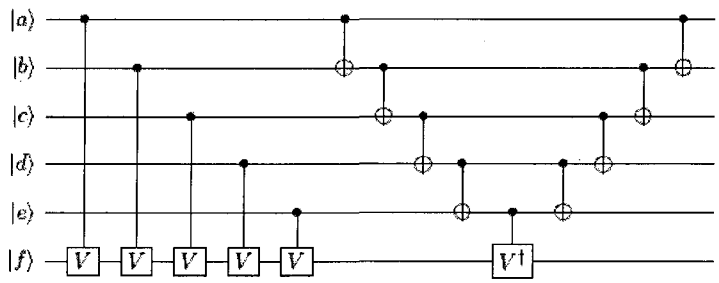


Figure 7.4.1: Realization of $S^{2,3}$ (a, b, c, d, e) ⊕ f using ARNGs. Observe the general pattern of connections.

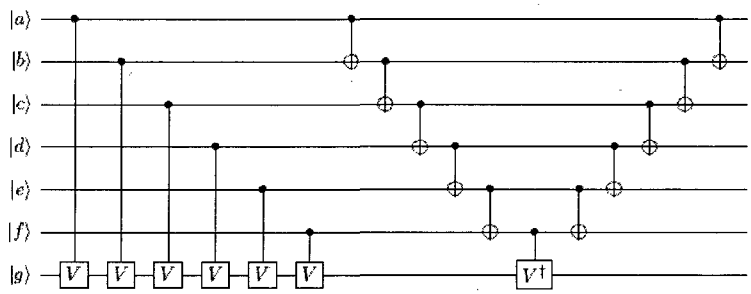


Figure 7.4.2: Realization of $S^{2,3,6}$ (a, b, c, d, e, f) ⊕ g using ARNGs. Observe the general pattern of connections.

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

(a)

ab \ c	0	1
00	I	V^+
01	V^+	I
11	I	V^+
10	V^+	I

(b)

ab \ c	0	1
00	I	I
01	I	NOT
11	NOT	NOT
10	I	NOT

(c)

Figure 7.4.3: KMaps for the lowest qubit of the circuit from Figure 7.2.7. (a) For controlled V, (b) For controlled $V^\dagger$, (c) the result of the composition of quantum maps 7.4.3a and 7.4.3b for the entire circuit from Figure 7.2.7.

Each, a, b and c contribute V's in KMaps (Figure 7.4.3a). Combining them with $V^\dagger$ from linear control, we get majority function (Figure 7.4.3c). Now, we can create those patterns for any number of qubits to get cheap realizations. If we would realize

these functions using standard multi-output Toffoli gates and next macro-generate them to 2×2 quantum primitives the cost would be very high. These 2-interval functions we call "cheap functions" because we use only CNOTs, CVs and $CV^{\dagger}$s here, and we achieve this design only by controlling single gates. Whenever we have to control with more than two controls, the circuit becomes more expensive, we have to add mirrors, sometimes ancillas and so on. So this gate construction method produces very cheap gates, we showed here that all these gates are cheap although they look more complex than Toffoli gates in KMaps. This is shown for three other 2-interval functions in Figures 7.2.10, 7.4.1 and 7.4.2. So, if we have a complex synthesis problem with many inputs, if we find any of theses gates be useful in the circuit, this will be the cheap part of the circuit. Recursive formula can be derived comparing those functions in formal way using my examples in Figures 7.2.10, 7.4.1 and 7.4.2. Unfortunately from these functions, we can not build every possible logic function, we need some other gates. But this idea was a good beginning.

Now, we will do the following. One generalization will be to take all possible linear functions, or affine functions as controls, this is the topic of section 7.5. Then we found that, it is even more interesting when these control functions were reversible not only linear. That would be another generalization. But before we started working with these controls which are not affine, we were still using affine functions but in more complex ways.

A possible approach is to implement a software simulator of these structures, one should have some kind of scripting methods that will generate all these problems automatically in a smart way. Again, this is a new topic, solved by nobody before, how to build the above presented type control which is only linear or affine, in the most efficient way. For instance, we may simulate all functions which will be created in the above Figures by replacing target V and V^+ gates in all possible ways by gates V and V^+. Functions obtained this way are all permutative and they can be all used as cheap functions in our synthesis methods.

Controlling with affine functions is always doable with no ancilla bits, because it uses only CNOT and NOT gates which can be next mirrored after using them to control some subcircuits, in the same collecting qubit to restore the original value of the function (such as an input). For instance, whatever the affine control, like $a \oplus b \oplus c'$, one can create this control signal "in place" (with no ancilla bits) using only CNOT and NOT gates. And next we can always concatenate mirror circuits, thus restoring the original inputs a, b, c. This is one more strong argument for affine gates – the simplicity of mirroring.

In addition, each linear function can be negated. We can substitute 4^{th} order square roots of NOT in place of V, V^+. We can systematically build gates of the types from Figures 7.2.10, 7.4.1 and 7.4.2 using any order roots of NOT. What is the difference with the V/ V^+ circuits? Now we can rotate in the target qubit by half smaller angles,

etc. Therefore, we can prove that we can build any gate but we use always half of the angles. There is a danger of using this method in some quantum technologies. Because, if we have very many input variables to the array, these angles will be very small, may be it will be susceptible to noise or decoherence.

Now the open question is, "Should we build Toffoli gates with restricting the angles, or should we rather add ancilla bits?"

The problem formulation is: we want to add minimum ancilla bits and restrict angles may be only to 90, 45 and 22.5 degrees, then we are able to build every Toffoli gate. The possibility of doing this was demonstrated in examples above. But how to do this best for each function is an open problem and is technology-related.

But now, when we have made this decision, we can analyze the cheap functions for these constraints. However the problem to be solved complicates, as when we want to realize arbitrary symmetric functions we have very many ways to combine the above two approaches and many choices of selecting the rotation angles. Maslov published a paper recently [Maslov07], he only proved the heuristic method for symmetric functions with standard k-input Toffoli gates. He does not take into account the inexpensive functions and the quantum realization aspects discussed here.

When one uses standard multi-input Toffoli gates, one either has to use the method from previous sections to make it quantum realizable or one needs to add ancilla bits. The methods presented here can find less expensive quantum realizations for several symmetric functions of few variables. But we still do not present a method to synthesize arbitrary symmetric functions to be realized systematically with interval gates from Table 7.2.1 and other affine gates. Approaches to solve this problem will be discussed in the sequel but the general problem is left unsolved in the thesis.

Here is some helpful theorem.

Theorem 7.4.1:

Any binary symmetric function can be built by Exoring a subset of symmetric base functions.

Proof.

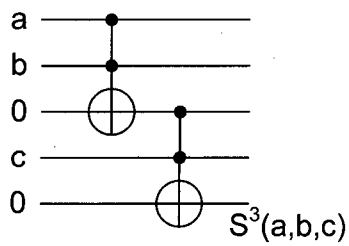
It can be easily proved that $S^{U \oplus V} = S^{U} \oplus S^{V}$. The idea would be thus to realize all possible symmetric functions S^{X} as base functions and calculate the quantum cost of each of these base functions. All orthogonal bases can be then created from these base functions and their matrices can be created. Next every symmetric function can be decomposed in each base and the total cost can be calculated for this base. Repeating this calculation for each base will give the minimal solution (these methods are

illustrated in detail in chapter 9 for general functions but symmetric base functions are just a useful special case).

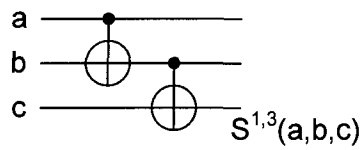
Example 7.4.1:

In case of 3 variable functions, the following symmetric functions are inexpensive base functions: $S^{1}(a,b,c)$ from Figure 7.4.4a, $S^{1,3}(a,b,c)$ from Figure 7.4.4b (a linear function), $S^{2,3}(a,b,c)$ from Figure 7.4.4c (an Affine Root of Not Gate). Creation of single-index function $S^{2}(a,b,c) = S^{\{2,3\}\oplus\{3\}}(a,b,c) = S^{2,3}(a,b,c) \oplus S^{3}(a,b,c)$ is shown in Figure 7.4.4d. Creation of single-index function $S^{1}(a,b,c) = S^{\{1,3\}\oplus\{3\}}(a,b,c) = S^{1,3}(a,b,c) \oplus S^{3}(a,b,c)$ is shown in Figure 7.4.4e. Creation of double-index function $S^{1,2}(a,b,c) = S^{\{1,3\}\oplus\{2,3\}}(a,b,c) = S^{1,3}(a,b,c) \oplus S^{2,3}(a,b,c)$ is shown in Figure 7.4.4g. Explanation of the composition using a Kmap is given in Figure 7.4.4f. Using the transformations from Figure 7.4.4h the circuit is finally optimized to the form from Figure 7.4.4i. As we see, all single-index and double-index symmetric functions of three variables are inexpensively realized with our methods (remember the macro-generation to CV and affine gates for products of variables). Similarly function $S^{1,2,3}(a,b,c)$ and other triple-index symmetric functions are inexpensively realized.

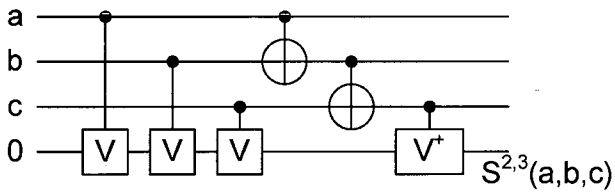
(a)



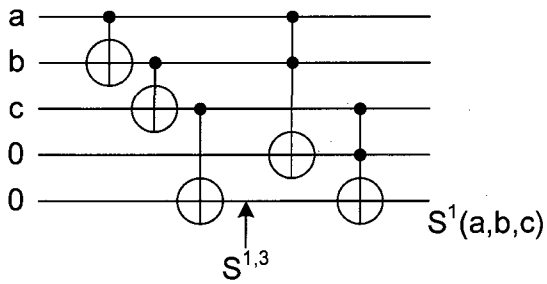
(b)



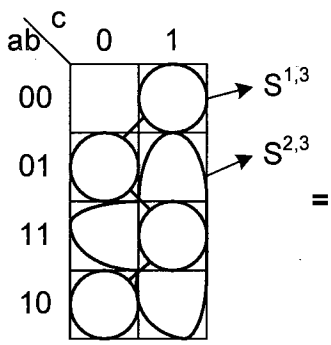
(c)



(e)



(f)



=

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

$S^{1,2} = S^{1,3} \oplus S^{2,3}$

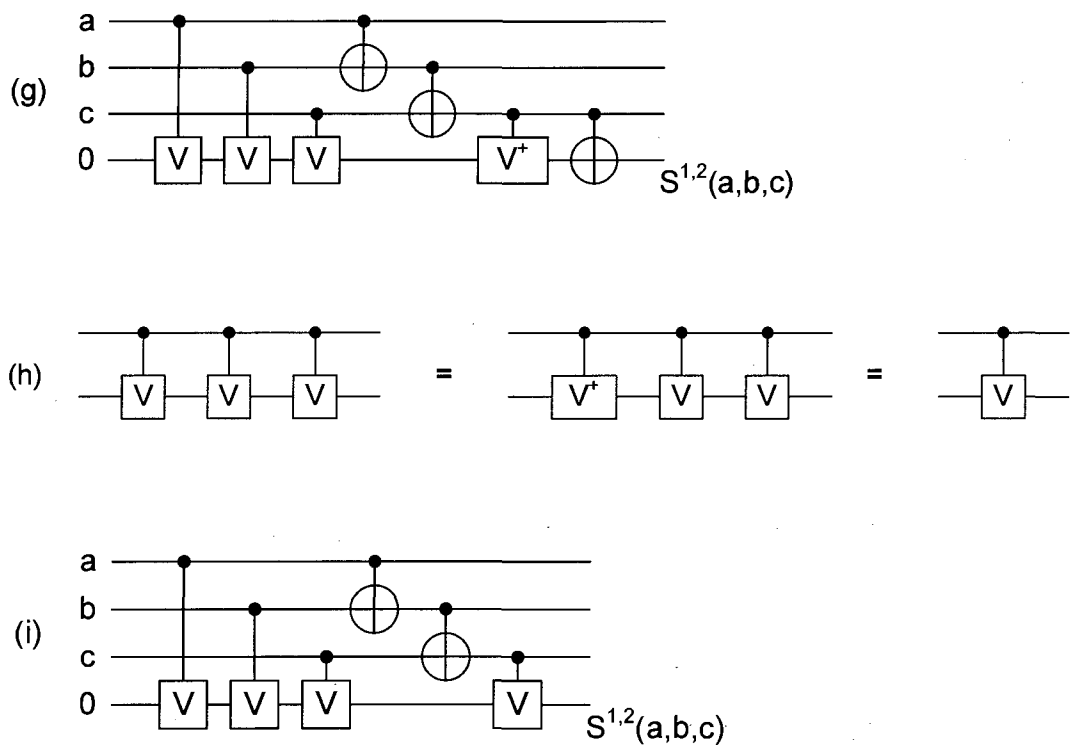


Figure 7.4.4: Synthesis of symmetric base functions and symmetric index-functions to illustrate the concept of symmetric bases. (a) Function $S^{1}(a,b,c)$, (b) Linear function $S^{1,3}(a,b,c)$, (c) an Affine Root of Not Gate function $S^{2,3}(a,b,c)$, (d) Creation of single-index function $S^{2}(a,b,c)$, (e) Creation of single-index function $S^{1}(a,b,c)$, (f) Kmaps to analyze the method, (g) Function $S^{1,2}(a,b,c)$ created by EXORing base functions, (h) auxiliary equivalence transforms, (i) Optimized realization of function $S^{1,2}(a,b,c)$ based on applying transformations from (h) and representing CNOT as CV•CV..

When we try to extend this method to functions of four and next five variables we see that the realizations of not all symmetric functions using our method are cheaper than the solutions from Maslov. However, a significant fraction of symmetric functions has smaller quantum costs than in Maslov's designs. Thus, in the worst case one may use the method from Maslov never obtaining worse results.

The presented here research on symmetric functions will also be very useful to create gates for MMD algorithm [Miller03] or arbitrary other algorithm for quantum synthesis, because we can create the inexpensive symmetric gates for any circuit width. We can build arbitrary quantum functions from these gates and using the methods from chapters 7, 8 and 9. Observe that in the standard ESOP minimization we use only Toffoli-like gates, but we see that in quantum we have all these majority gates, 2-interval gates and other gates which are all very cheap gates. Nobody has proposed this idea of using other gates in the framework of ESOP synthesis so far, because problems like this did not exist in classical logic.

When a reversible function is to be realized, every output of it is a balanced function which has equal number of ones and zeros. This property is extremely useful to limit the search. Every reversible gate like Toffoli, Miller, Fredkin has the property that every output function of each of these gate is balanced: half zero's and half ones. This property immediately decreases the space of search very much. Also any kind of symmetry limits the search extremely. In our basic Barenco-extended circuits, with V and $V^{\dagger}$, if we do every possible permutation between V and $V^{\dagger}$ like a binary order, VVVV, VVVV†, VVV†V, ..., V†V†V†V† each of them will generate some new gate. Because we randomly combine these gates, we create many gates that are not permutative, as we have single V, then it will have half probability of ones and half zeros. Always we create a new design and the program CircuitSearch will verify if the created circuit is permutative. This approach is based on analysis, first our search

method was naïve. Based on analysis and generalization we created next systematically new improved library of gates for small numbers of variables. The hierarchical design methods of blocks shown in Chapter 11 demonstrate the practical use of such gates.

If the single-output function is balanced then we can realize the function directly and with no ancilla bits. Our methods generalize therefore the Maslov's method [Maslov07]. Although Maslov deals only with Toffoli gates and we deal with many types of controlled gates, the properties of layers of the quantum circuits are very similar. A new definition of symmetry is possible. If we substitute in the structure the controlled-V and controlled-$V^{\dagger}$ gates in all possible ways, we will obtain many quantum gates. So we can now introduce the concepts of the quantum circuit symmetry. By introducing the rotation here, we will introduce many quantum functions, some of them will be binary, other will be not binary. Everything that we invent here is basically a generalization of classical binary symmetry. Now we have to use these symmetrics in quantum circuits with minimum number of ancilla bits and use also Toffoli gates with the minimum number of inputs. Every component of this function can be reused to build other function. These general ideas are detalized and illustrated in next sections.

7.4.2. Conclusions on 2-interval and symmetric functions.

We recall Toffoli gate circuit from Figure 7.2.1, basically many ideas of my dissertation start from this circuit [Barenco95]. V is the square root of NOT. Most important that we can change the V gate from square root of NOT to the 4^{th} order root of NOT, the 8^{th} order root of NOT and so on. Again NOT . NOT = I (identity or wire or same as before or can be cancelled). The rules are V.V = NOT, NOT.NOT = $V.V^{\dagger}$ = I (means identity or wire or goes through or can be cancelled or omitted). Next the rules are G.G = V, $G.G^{\dagger}$ = I and so on. The idea occurred therefore to create the software which will generalize all these quantum identities for arbitrary root gates, extend for more inputs, etc, keeping the structure.

Figure 7.2.1 is just one example; our exhaustive search program CircuitSearch generates all such combinations or structures. When the program works for 3 inputs from which a, b are the control bits and c is the controlled bit, we search for all possible affine function with V, $V^{\dagger}$ and NOT.

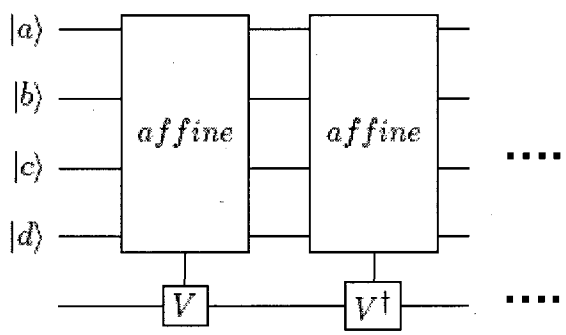


Figure 7.4.5: Example of a structure with affine controls of $V/V^{\dagger}$ gates.

Our generalization will be here to take all possible linear functions (or affine functions) or some subsets of them, randomly generated as the controls of root gates in the target qubit. See an example in Figure 7.4.5. This is a new approach to synthesis again. Instead of checking by hand and trying to prove facts to invent new useful gates for synthesis, I decided to write a simulator/generator to help me in this analysis.

Observing next all these new circuits generated by my program CircuitSearch, I found many new circuits and more importantly we got new circuit realization ideas. The circuit types from chapters 3, 7 were generated and more generalized circuits and gates from chapters 8 and especially 9 were next generated. This program, a fast prototyping tool, stimulated much my mind. One can appreciate that all controls are linear, affine or reversible (balanced) functions. Thus all controls in our basic blocks will be of these types.

As we see above, the principle of our approach is very simple. Knowing a powerful pattern of creating circuits from this chapter, we use this pattern to systematically or stochastically generate new gate families of interesting gates. In our first variant of the generator we have all affine functions as control functions and we use V, $V^{\dagger}$ and NOT in the data path. It is next relatively easy to generalize this approach using three methods:

1) generate non-affine controls in variables a, b and c to generate such circuits.

2) add ancilla bits,

3) extend the set of root gates in the target qubit.

7.5. The Program Generator to Synthesize Quantum Arrays with "Affine Root of NOT" Gates

7.5.1. Introductory ideas

My idea at first was to allow my computer to spend much time, even days and weeks, to find the exact minimum solutions (to useful gate) and next to use such "inventions" as higher level "building blocks" in quantum circuits. Exhaustive search [Lukac05, Miller04] has been already used before in reversible logic design, but there are many ways how the exhaustive search can be organized, and they differ in processing time and memory usage. We investigated several types of exhaustive search strategies applied to particular quantum circuit structures (chapter 6). We found that for this kind of problems the A* algorithm known from AI [Nilsson71] operates very similarly to the breadth first search. Our IDDFS [Giesecke07] search is similar but it is easy to program and uses less memory, thus allowing to minimize larger circuits. In chapter 6 we proposed even more general search strategies that I used already for other problems (chapter 8). We use search strategies also in this chapter.

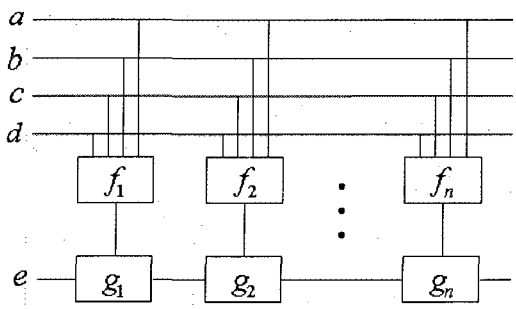


Figure 7.5.1.1: Generalized structure to explain the operation of the CircuitSearch generator program.

Basic explanation of our software follows. The CircuitSearch program creates (as per our specification that means using affine control function and taking all possible combination of V, $V^{\dagger}$ and NOT, it can be with all V's or combinations) one possible circuit (next its function) for 3 input variables (a, b, c) as Figure 7.5.1.2. In Figure 7.5.1.2 all inputs a, b, c are the same as the outputs A, B, C. That means, in the program a, b, c lines are only for activation of gates in the target line. At first, we only care about single output function f.

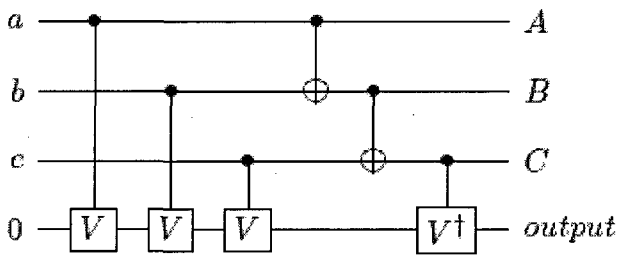


Figure 7.5.1.2: The circuit given to test our program CircuitSearch. The truth table of this permutative circuit is the program's input.

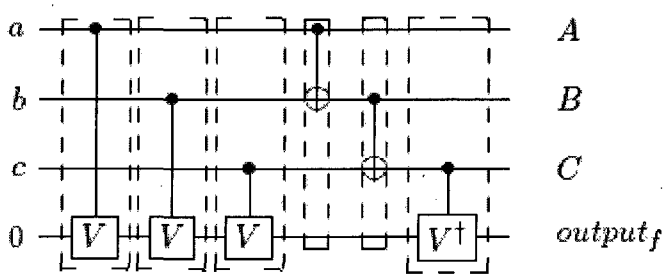


Figure 7.5.1.3: Partitioning of the quantum circuit from Figure 7.5.1.2 for Genetic Algorithm used by previous authors.

Now let us present the inner loop of our program. Here the program verifies the generated by it circuit comparing its truth table with the table of a specification function, as explained below. This can be a binary KMap, it can be a truth table as well. Program will compare the stored KMap (the binary function) with this function generated and simulated from the circuit's structure, cell by cell. If the two functions completely match in all 2^n cells, then the program declares that it found such circuit after exhaustive search. Below KMap is for the inner loop of the program. In this case the program will say that after exhaustive search it found this data (Figure 7.5.1.5). It is for 3 input variables. Program works for 4, 5 and as many as possible input variables. As CircuitSearch is memory intensive, how many input variables are possible depends on the problem size. I tried to use my program for the maximum number of variables. Thus, we have two loops in our program, outer loop will create all binary functions (as their circuits) using exhaustive search and our problem-defining methodology and constraints specification (with all possible affine functions of input variables and applying all possible V, $V^\dagger$ and NOT in the target qubit). The program's inner loop compares the circuit found with the function specification in the

form of a truth table or a KMap. With this specific function, my program verifies whether this specific function is generated by the program by comparing all minterms.

Suppose one wants to use the CircuitSearch program to create the structure of f = majority (a, b, c) $\oplus$ 0 = majority (a, b, c). The circuit from Figure 7.5.1.2 is expected to be found. However, because of the way how circuits are partitioned in our generator (Figure 7.5.1.1), the circuit is not partitioned as for Genetic Algorithm (Figure 7.5.1.3) but it is partitioned as in Figure 7.5.1.4. It is more efficient.

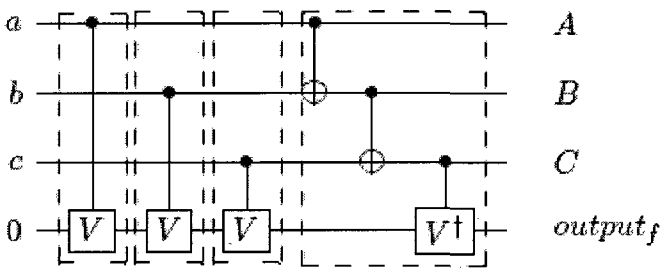


Figure 7.5.1.4: An example of created circuit for 4 segments, $a \oplus b \oplus c$ is one possible affine function from Figure 7.5.1.3 but generated directly for a single control, found by my program.

This is a circuit that contains 3 inputs and 4 segments. The first segment has the control "a" and the target 0 as inputs. While the fourth segment takes $a \oplus b \oplus c$ as its control input. One possible circuit for 3 input variables (a, b, c) is presented in Figure 7.5.1.2. In this Figure, all inputs a, b, c are the same as outputs A, B, C. That means, in the program a, b, c qubits are only for the activation of the target qubit.

The only care is given to output f. The program calculates the KMap for output f. If any non binary value shows up in the simulated symbolically QMap output f, such as any single value like V, $V^{\dagger}$ in the QMap, then the output f is not binary. Such output is not useful as we synthesize only permutative circuits. The "Affine CircuitSearch" system omits those non binary outputs and searches for the next possible circuit which will hopefully correspond to the binary specification function.

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

Figure 7.5.1.5: Example KMap Specification of binary values in Affine Circuit Search method for the target qubit f from Figure 7.5.1.2.

Hence the specification as in Figure 7.5.1.5 is finally matched in every cell, of course if sufficient time and memory space is allotted to CircuitSearch.

7.5.2. Reduction of circuits to binary

Reduction of general quantum circuits to binary circuits is done according to the following rules:

CircuitSearch uses only V, $V^{\dagger}$, NOT gates with affine function. It uses the following formulas:

a) V = square root of NOT,

b) $V \odot V = NOT$

c) $V \odot V^{\dagger} = I$

d) $NOT \odot NOT = I$

A step by step example:

Example 7.5.2.1:

Figure 7.5.1.2 shows a typical quantum circuit that can be found by the program. It is explained below how Quantum Map rules are used to calculate the final QMap of the circuit model shown in Figure 7.5.1.4. Inputs a, b, c control the gates. So that when the control input to the controlled gate is 1, then that gate becomes active. Input d is assumed to be a 0.

1. The QMap for input a controlling a V gate is shown in Figure 7.5.2.1.

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

Figure 7.5.2.1: QMap 1 (symbolic) for V controlled by input a in circuit from Figure 7.5.1.4.

2. The QMap for b controlling a V gate is shown in Figure 7.5.2.2.

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

Figure 7.5.2.2: QMap 2 for V controlled by input b.

3. The QMap for the V gate controlled by input qubit c is shown in Figure 7.5.2.3.

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

Figure 7.5.2.3: QMap 3 for V controlled by input c.

4. The QMap for combined QMaps 1, 2, 3 for Figure 7.5.1.4 is shown in Figure 7.5.2.4.

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

Figure 7.5.2.4: The combined QMap for 3 V's controlled by inputs a, b and c each.

5. For the fourth gate: QMap for a⊕b ⊕ c becomes KMap for a⊕b⊕c, Figure 7.5.2.5.

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

=

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

Figure 7.5.2.5: QMap for $a \oplus b \oplus c$ is a KMap.

6. QMap for $(a \oplus b \oplus c)$ controlling $V^{\dagger}$ gate is shown in Figure 7.5.2.6.

ab \ c	0	1
00	0	V^+
01	V^+	0
11	0	V^+
10	V^+	0

Figure 7.5.2.6: The QMap of $V^{\dagger}$ controlled by control function $a \oplus b \oplus c$.

7. QMap 6 and QMap 4 are combined to become QMap 7, as shown in Figure 7.5.2.7.

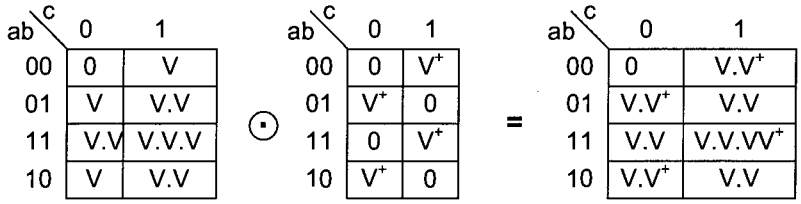


Figure 7.5.2.7: Combining QMaps with composition operator for the entire circuit from Figure 7.5.1.4.

8. Using our formulas the QMap can be reduced as in Figure 7.5.2.8.

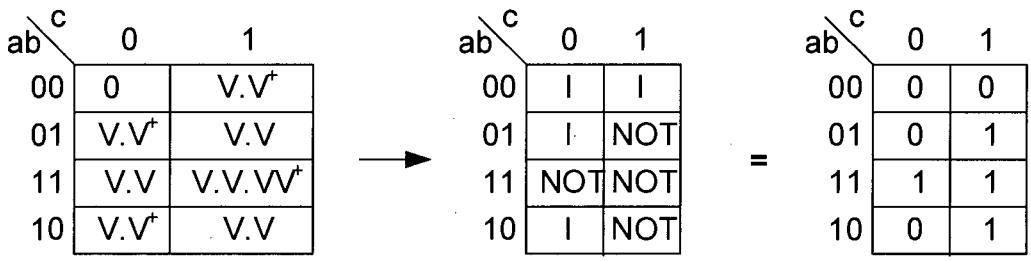


Figure 7.5.2.8: Reduction of the symbolic QMap to the standard KMap of the function realized by the exhaustively generated circuit. I = I (d) = d = 0 and NOT = NOT (d) = NOT (0) = 1.

1. Since the final QMap has value 0s, 1s, I (identity) and NOT. Then this circuit is a binary circuit. It is accepted by the program as the solution to the formulated specification function from Figure 7.5.1.5. It can be printed as soon as it is found. If the search is completed for all circuits within given constraints, then we know that this solution is the exact minimum.

The current system is intended to generate all possible QMaps using the exhaustive search.

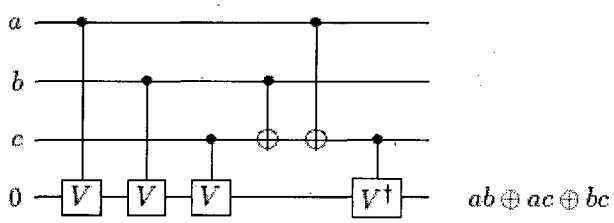
7.6. Using Cheap Quantum Gates (CQG) in general AND/EXOR synthesis.

7.6.1. From Affine Root of NOT Gates to Affine Toffoli Gates and Affine Complex Gates.

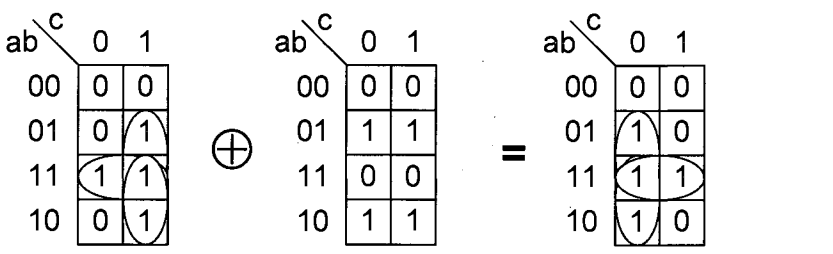
The cheap gates are based on symmetric composition of CV and $CV^{\dagger}$ gates. Another cheap gates realize affine functions. Compositions of these two types of inexpensive gates allow to realize other functions with reduced costs. This section discusses some of the composition and universality problems.

Example 7.6.1.1:

a)



b)



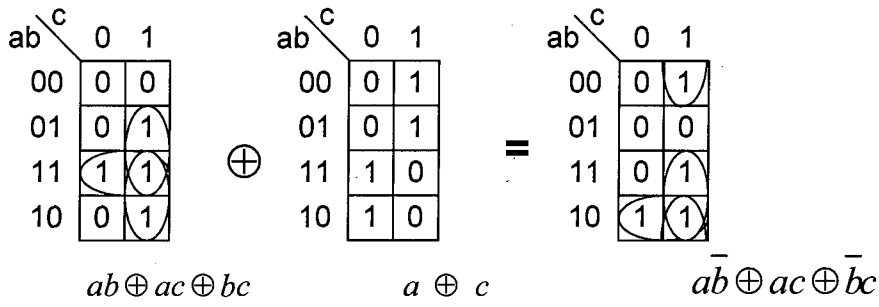


Figure 7.6.1.1: Re-use of the basic majority pattern: (a) The function of the basic circuit with target qubit set to 0, (b) exoring the basic majority with another cheap function, linear $a \oplus b$ function, leads to another majority, (c) exoring one cheap function leads to another cheap function being one more majority function (polarity shift only).

Now, we know that the realization of the three-input majority is cheap and the realization of CNOT is also cheap, we ask ourselves the question "what other functions can be inexpensively realized by combining these two types of gates?" Figure 7.6.1.1b shows that by EXORing with $a \oplus b$ we obtain another majority function, but this time with a different polarity. The same is true while EXORing with $a \oplus c$ - Figure 7.6.1.1c.

However, as illustrated in Figure 7.6.1.2, when EXORing with variable c we obtain a new pattern of dual minterm functions known to be difficult to realize in AND-EXOR logic (see chapter 3). Similarly, the realization of the majority functions with all their possible polarities is cheap (Figure 7.6.1.1). Other dual minterm functions (called also minterm pair) for 3 variables are shown in Figure 7.6.1.3 a,b.

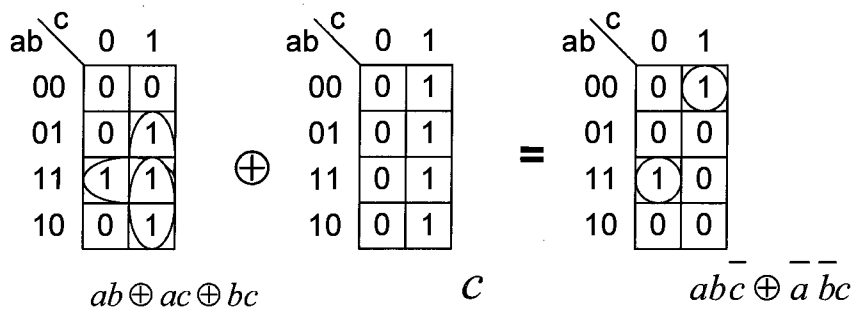
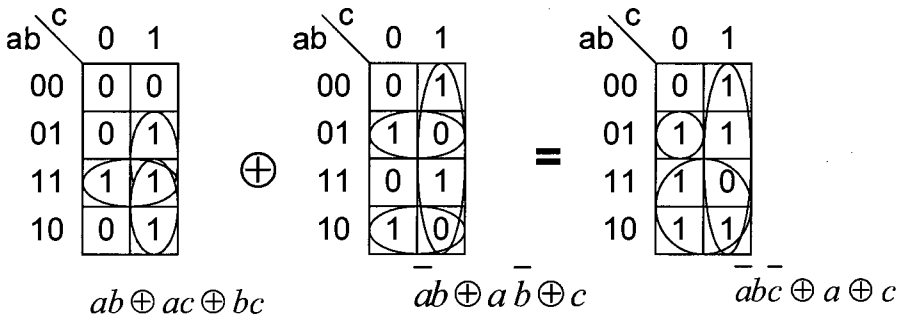
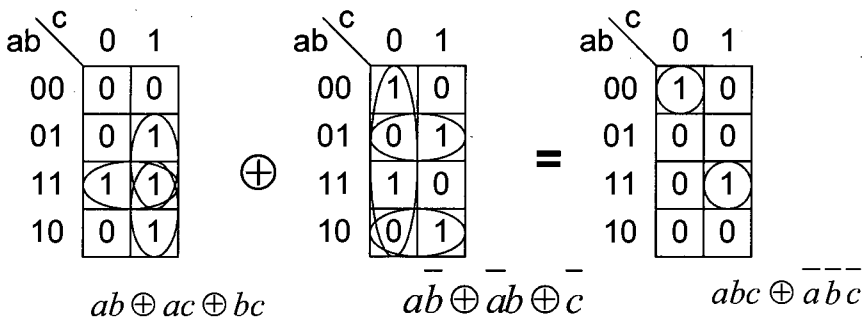


Figure 7.6.1.2: Shows that by exoring with variables we create dual-minterm functions of Hamming distance 3.



a)



b)

Figure 7.6.1.3: Exoring the cheap functions. a) Presents that by exoring with a linear function of 3 variables we obtain the negation of the dual function $\bar{a}\bar{b}\bar{c} \oplus abc$ *. b) Shows that exoring with the affine function* $a \oplus b \oplus c \oplus 1$ *we obtain the dual-minterm function* $\bar{a}\bar{b}\bar{c} \oplus abc$ *.*

Concluding, by combining all ARNGs of 3 variables with all affine functions we can create all dual-minterm functions. Therefore every even function of 3 variables, i.e. a

function having an even number of minterms can be realized with reduced price using our approach. If the function to be realized is odd, then all its minterms but one are realized using this approach, so the improvement is also substantial. The remaining minterm (full product of all variable literals) has to be however realized as a standard 3×3 Toffoli gate, which is expensive. This needs to be done however only for one minterm, so in general only one multi-input Toffoli gate of the highest complexity is used in the entire circuit.

Example 7.6.1.2:

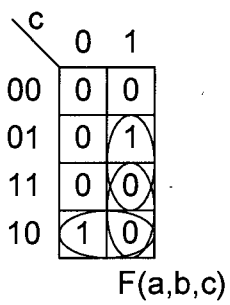


Figure 7.6.1.4: The Even HD3 function to be synthesized in Example 7.6.1.2.

Given is function F(a,b,c) from Figure 7.6.1.4. As we see, this is an even function as it has 2 true minterms. It is also a "minterm pair" function of HD = 3. Thus we expect that there is an inexpensive realization of this function. Using standard AND/EXOR logic we obtain GRM $F = bc \oplus ac \oplus a\bar{b} = (a \oplus b)c \oplus a\bar{b}$ which leads to the realization from Figure 7.6.1.5.

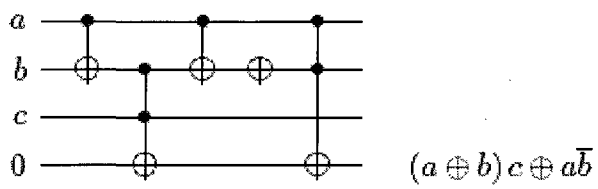


Figure 7.6.1.5: Standard method to realize the function from Figure 7.6.1.4. It uses GRM and factorization.

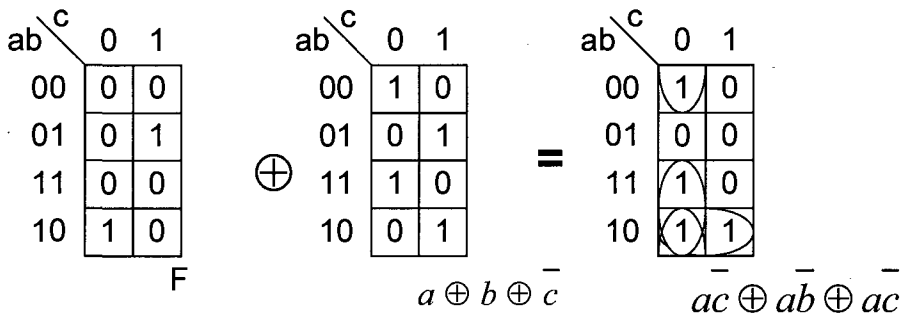


Figure 7.6.1.6: Analysis to be used in our new method to realize the function from Figure 7.6.1.4. Because $F \oplus (a \oplus b \oplus \bar{c}) = maj(a, \bar{b}, \bar{c})$ *then* $F = maj(a, \bar{b}, \bar{c}) \oplus (a \oplus b \oplus \bar{c})$.

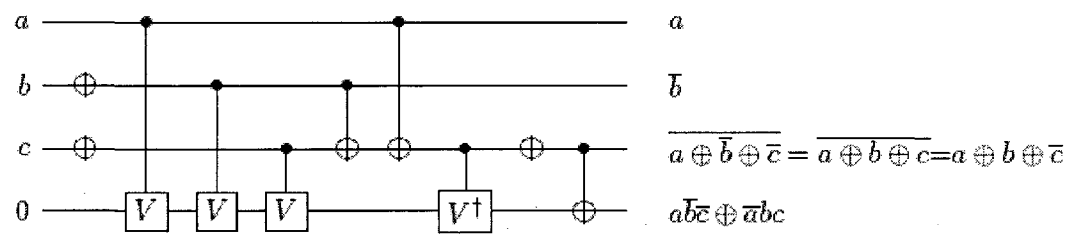


Figure 7.6.1.7: Quantum Circuit for F based on equation $F \oplus (a \oplus b \oplus \bar{c}) = maj(a, \bar{b}, \bar{c})$.

The solution from Figure 7.6.1.7 requires seven 2×2 gates and three inverters. The solution from Figure 7.6.1.5 requires two 3×3 Toffoli, two CNOT and one inverter, which means (2×5 + 2) = 12 2×2 gates and one inverter. Both these solutions are clearly better than the direct circuit realization with two 4×4 Toffoli gates. This suggests that every even function with 2 r minterms can be represented by exoring r dual-minterm function. Every odd function can be realized as an EXOR of a minterm and an even function. Therefore, our method improves the cost of every Boolean function. (We discussed only the single-output case so far). The principle is : "the function with 2r minterms should be partitioned to r "minterm pairs"."

We proved therefore the following theorem.

Theorem 7.6.1.1.

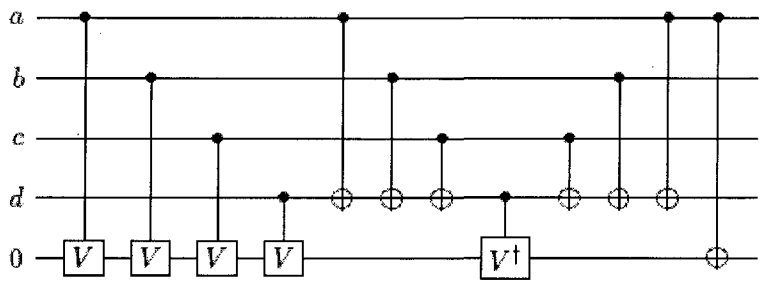
Every function of three variables that has more than one minterm can be realized with reduced cost using the introduced earlier ARNG gates.

Proof.

Every function is either even or odd. Every odd function of 3 variables that has more than one minterm can be decomposed to an even function and a minterm.

This property is also true for even functions of arbitrary number of argument variables. To prove this fact let us first consider functions of four variables.

(a)



(b)

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

⊕

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

=

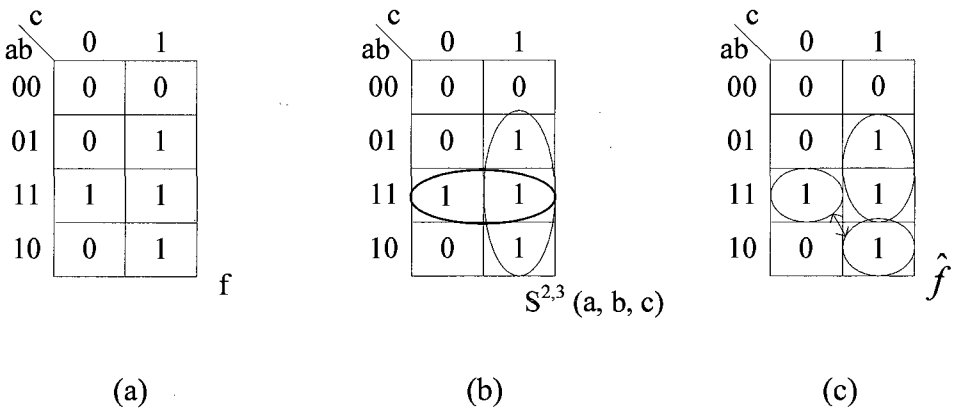
ab \ cd	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	0	1	0
10	1	0	0	0

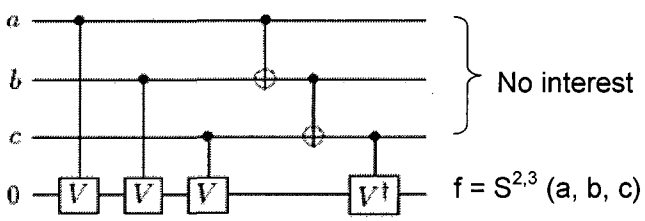
Figure 7.6.1.8: Function $S^{2,3}(a, b, c, d) \oplus a$. (a) The Circuit (b) KMap analysis of this circuit.

The even function realized with ARNGs always brings gain and larger groups are always better (Figure 7.6.1.9, Figure 7.6.1.10).

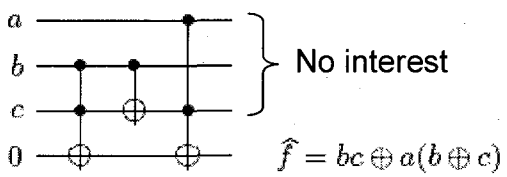
Boolean function for the circuit from Figure 7.6.1.8 can be calculated from composition as $S^{2,3}(a, b, c, d) \oplus a$. Figure 7.6.1.8b shows the analysis of this function.

Example 7.6.1.3:





(d)



(e)

Figure 7.6.1.9: For function f from Figure 7.6.1.9a the symmetric grouping is shown in Figure 7.6.1.9b, while a non symmetric grouping is shown in Figure 7.6.1.9c. The grouping from Figure 7.6.1.9b is realized in Figure 7.6.1.9d while the grouping from Figure 7.6.1.9c is realized in Figure 7.6.1.9e.

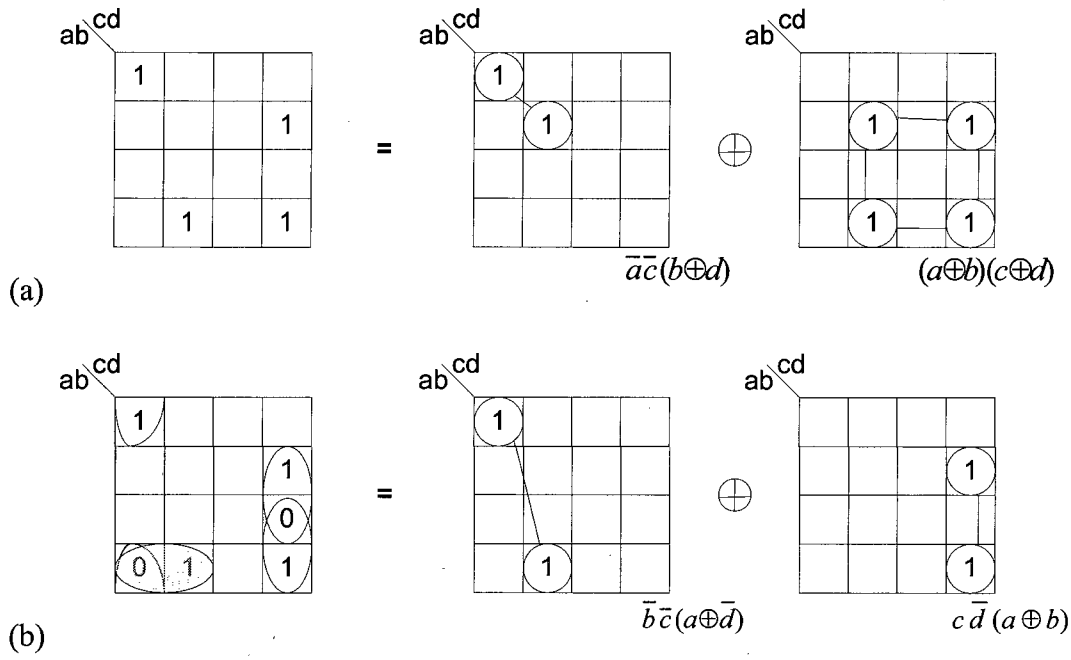
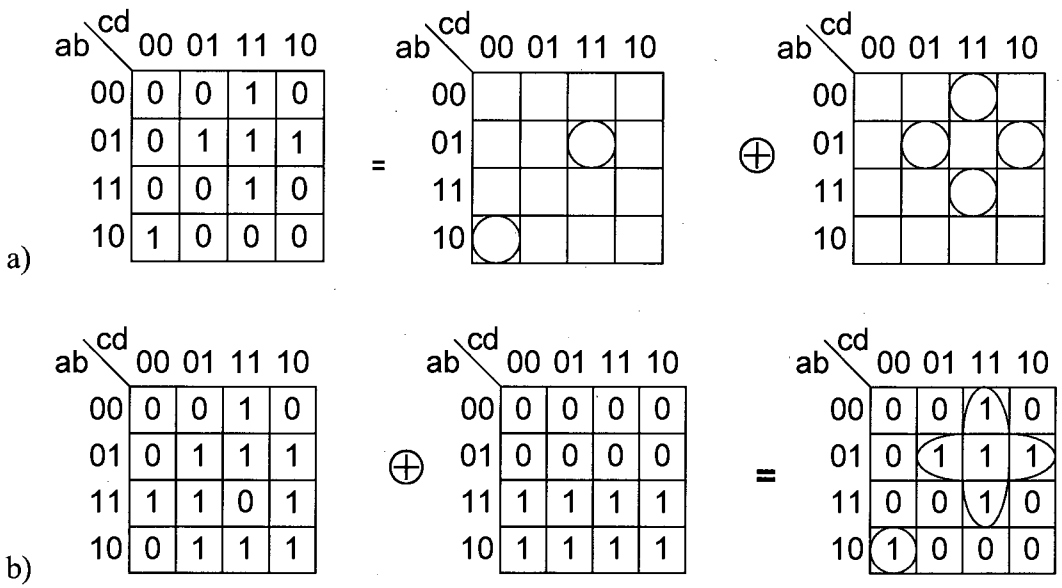
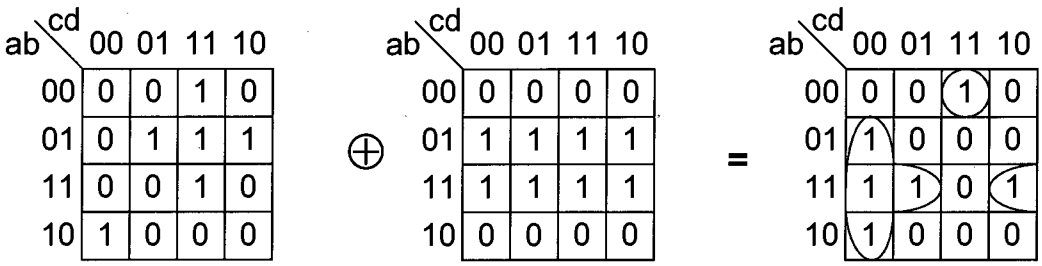


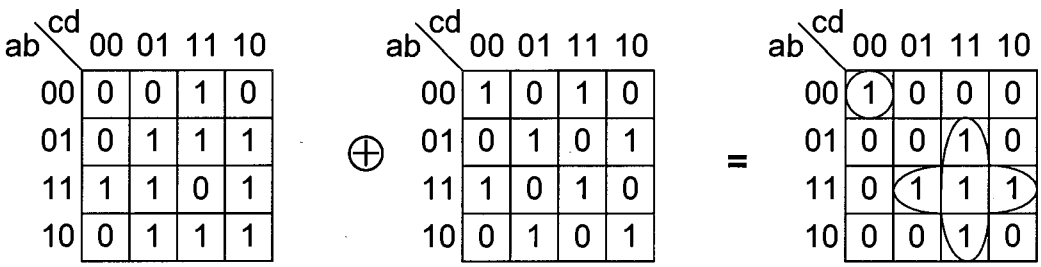
Figure 7.6.1.10: Realizing bigger groups is always better. (a) decomposition to 2^1-minterm group and a 2^2-minterm group, (b) decomposition to two 2^1-minterm groups has a higher quantum cost.



c)



d)



e)

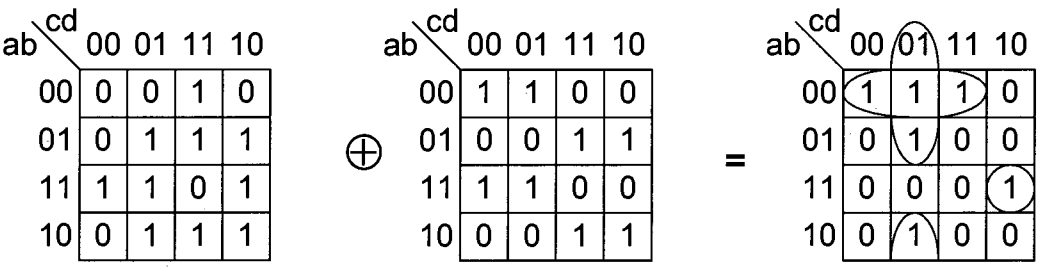
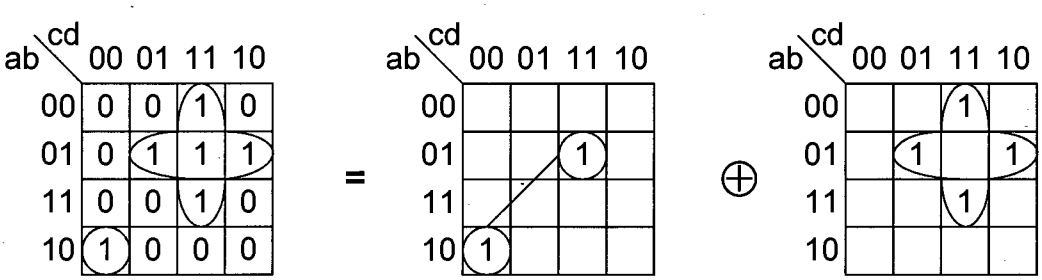
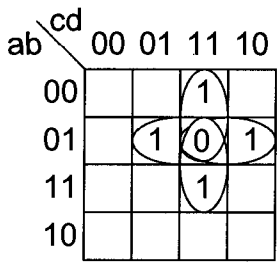


Figure 7.6.1.11: Examples of four variables functions that can be generated from 2-interval and affine functions.

a)

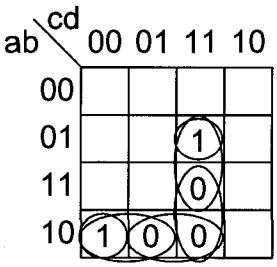


b)



$$\bar{a}cd \oplus bcd \oplus \bar{a}bd \oplus \bar{a}bc$$
$$= (\bar{a} \oplus b)cd \oplus \bar{a}b(c \oplus d)$$

c)



d)

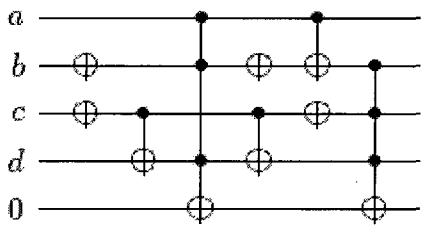
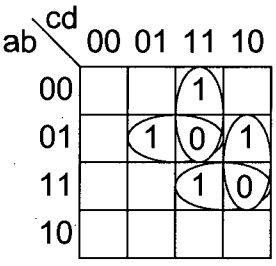


Figure 7.6.1.12: (a) EXOR decomposition of function from Figure 7.6.1.8. (b) S^3 ($\bar{a}$, b, c, d), (c) realization of HD4 function of 4 variables using the crosslink synthesis operator of cube calculus, (d) its realization.

a)



$$\bar{a}bd \oplus \bar{a}cd \oplus abc \oplus bc\bar{d}$$
$$= \bar{a}d(b \oplus c) \oplus bc(a \oplus \bar{d})$$

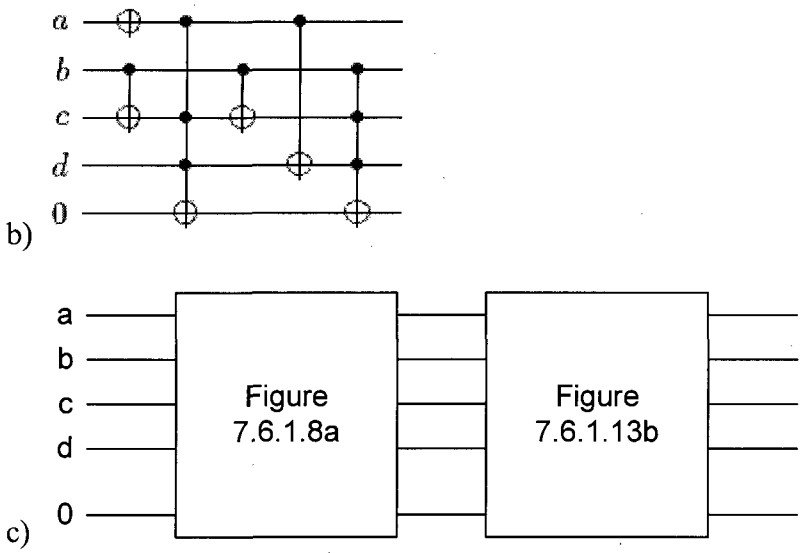


Figure 7.6.1.13: (a) S^3 ($\overline{a}$, b, c, d) and its factorized equation with Affine Toffoli gates, (b) corresponding quantum array, (c) realization of function from Figure 7.6.1.12c as a composition of inexpensive circuits.

Concluding. Figure 7.6.1.10 proved that any HD3 minterm pair function of 4 variables can be realized with reduced cost using affine Toffoli gates. Figure 7.6.1.12 proved that any HD4 minterm pair function of 4 variables can be realized with reduced cost using Affine Toffoli gates. Thus any minterm pair function of 4 variables has reduced cost.

Using this decomposition one can prove that every even function of 4 variables can be decomposed to a set of pair of minterms (with Hamming distances HD1, HD2, HD3, …. , etc) and often can be reduced to cheap affine and 2-interval functions of other types.

The function F with more than one minterm can be realized with affine gates with cost savings when compared to a solution of this function with multi-input Toffoli gates. The single-minterm functions can not be improved by using affine functions. While grouping 2^k minterms to affine functions, the symmetric realizations are always better.

7.7. Affine Polarities.

Affine preprocessor is any vector of affine functions. Every function can be realized into standard polarity preprocessor P_i, affine polarity preprocessor AP_i, PPRM, mirror of AP_i and mirror of P_i.

Algorithm 7.7.1:

For all polarities P_i do:

For all affine AP_j polarities do:

a) transform function F to $\overline{F}$ in this combined polarity $P_i \bullet A\ P_j$

b) calculate PPRM for $P_i \bullet A\ P_j$

c) realize the circuit of the polarity preprocessor P_i and its mirror post processor P_i^{-1}

d) realize affine polarity pre-processor $A\ P_j$ and its post-processor $A\ P_j^{-1}$, insert this pair between P_i and P_j ,

e) insert PPRM in the middle between $A\ P_j$ and $A\ P_j^{-1}$.

The same is true for every single gate, as shown in Figure 7.7.1 and Figure 7.7.2.

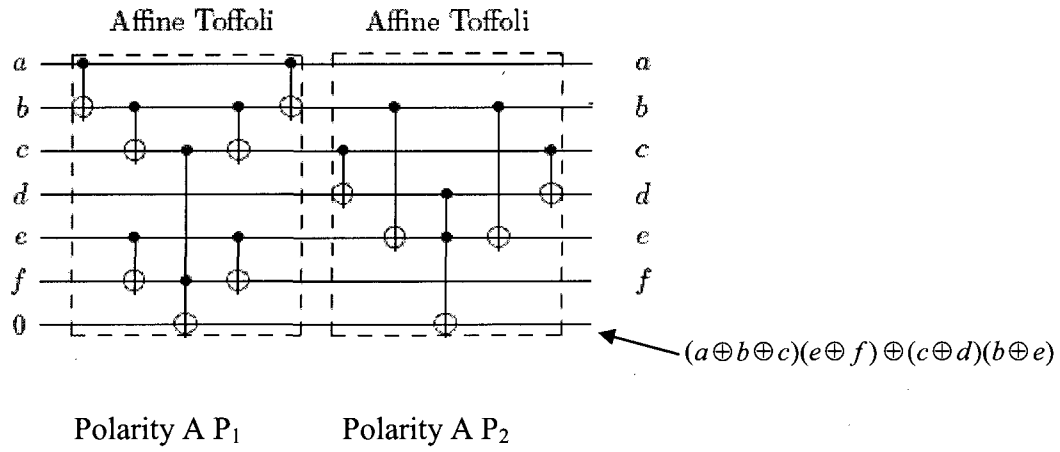


Figure 7.7.1: Oracle being a composition of two Affine Toffoli gates with different affine polaritie .

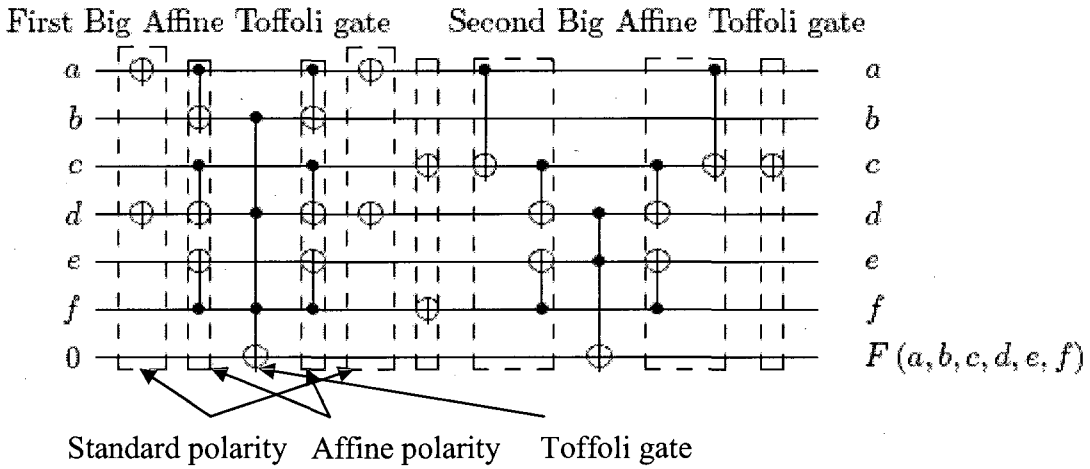


Figure 7.7.2: Realization of quantum arrays with affine gates realized according to Algorithm 7.7.1.

The idea of combining standard and affine polarities leads therefore to two new concepts:

a) Affine Toffoli gates, ARNG gates and other affine gates that can be used individually in synthesis methods (for instance to synthesize circuits that generalize ESOP).

b) The generalization of the concept of PPRM. A PPRM with a standard polarity preprocessor and postprocessor is an FPRM. Thus our new concept of new AND/EXOR family generalizes the FPRMs.

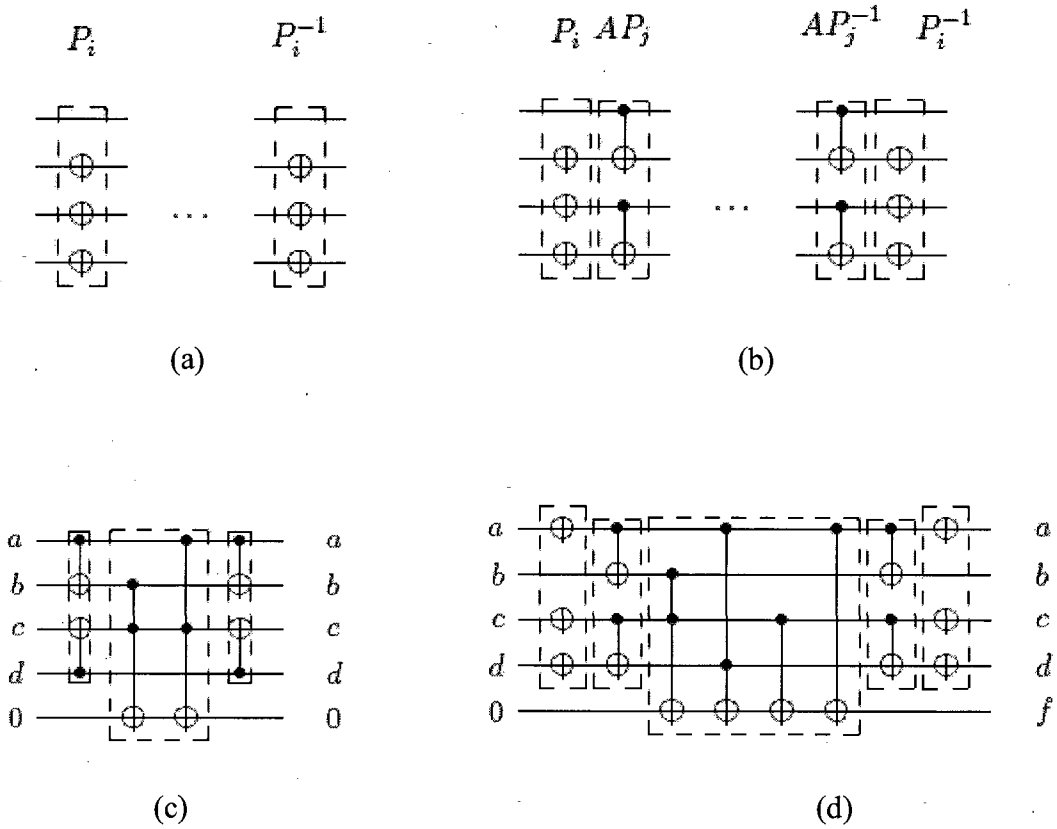


Figure 7.7.3: (a) Preprocessor and postprocessor for Standard polarities, (b) Pairs of the Preprocessor and postprocessor for arbitrary circuits, (c) example of simple linear affine preprocessor for a PPRM $bc \oplus ac$, (d) example of an FPRM generalization created by adding linear pre- and post- processors.

Figure 7.7.3 shows Standard polarity gates and affine polarity gates together with their mirror to create an oracle for kernel PPRM function $\hat{b}\hat{c} \oplus \hat{a}\hat{d} \oplus \hat{c} \oplus \hat{a} = (\overline{a} \oplus b)\overline{c} \oplus \overline{a}(\overline{c} \oplus \overline{d})\overline{c} \oplus \overline{a}$.

7.8. Program CircuitSearch

7.8.1. Introduction to CircuitSearch

CircuitSearch realizes a new approach to design quantum circuits using different search strategies. Here are some of its properties:

1. Visualization. The user can visualize the circuits. This helps the user in investigating new search algorithms and the solution space. This research is the first application of the visualization of circuits in the classical reversible and quantum forms (QMaps).

2. Exhaustive Search. A CircuitSearch algorithm finds solutions using the state-space search mechanisms. Human-designed expert systems often work well, but are limited in application. Traditional pure search strategies are comprehensive, but memory and time intensive. The heuristic search methods of Genetic Algorithms/Genetic Programs have limitations of size, computation time, and solution optimality and further, give no explanation of design methodology or transferable rules for generalization. Human expertise must therefore combine with search mechanisms, for the development of efficient

problem-solving methods. Thus the human can control and modify the CircuitSearch program.

3. Affine CircuitSearch represents a rich example for a problem that has a very large search space. I created a system that can enumerate logical circuits with specific characteristics and optionally matching function signatures (QMaps). It can use two different search techniques – exhaustive and iterative deepening – which are both blind (no heuristics are used). Using exhaustive search or iterative deepening search consumes a lot of resources. It takes longer CPU time and sometimes more memory (according to the implementation details). The user can interact and reduce the search.

In order to design a circuit that performs a desired quantum computation, it is necessary to find a decomposition of the unitary matrix that represents that computation in terms of a sequence of quantum gate operations. The initial search idea of our research is very naïve, we want to visualize the quantum circuit constructed from very basic quantum gates which comprise V, $V^{\dagger}$ and NOT gates. The purpose of CircuitSearch is to enumerate circuits using a variety of methods for the user that can control the search by additional parameters as a result of his visual inspection. To abstract this idea of searching, the CircuitGenerator interface is introduced. Different search methods can implement this interface, and the rest of the program, more or less, doesn't care how actually the search is working.

CircuitSearch is a C# .NET application, developed with Visual Studio 2005, and designed for enumerating logical circuits with specific characteristics and optionally matching function signatures (KMaps). It uses two different search techniques – exhaustive and iterative deepening – which are both blind (no heuristics are used).

7.8.2. Affine Circuit Search Implementation

The two main methods are Advance() and GenerateCircuit().

Advance(): steps the generator so the next circuit is evaluated. **GenerateCircuit():** will configure the passed Circuit object to reflect the current state of the generator.

GetProgress(): method can provide an estimation of the percentage of complete enumeration.

The front end program writes all binary circuits out to file for future use - either by the browser, or as a saved search to match against KMaps.

Three generators are currently implemented: ExhaustiveCircuitGenerator, CircuitDatabase, and IterativeDeepeningCircuitGenerator.

Description of Main Methods:

1. **ExhaustiveCircuitGenerator:**

   This is the constructor that is Responsible for creating the segment generators.

2. **GenerateCircuit:** Responsible for creating and configuring new circuit.

3. **Advance:** At each *Advance*() call, the generator advances the rightmost SegmentGenerator.
4. **SegmentGenerators Object:** represent the state of each segment.

Pseudo code:

This generator acts like a big counter. A list of SegmentGenerator objects are created. At each Advance() call, the generator advances the rightmost SegmentGenerator. If that is at its end, it advances the next SegmentGenerator in line and resets the first. This happens to the entire array iteratively. The enumeration is exhausted when all SegmentGenerators are at their end.

The SegmentGenerators represent the state of each segment: which inputs are sampled, whether the affine function includes negation, and which function is applied (V, $V^{\dagger}$ or NOT). Each advance call on the SegmentGenerator will 'tick' the state so that the next circuit is generated. First the negation is toggled, then the function is advanced, and then the input sample counter is advanced.

The following Figure 7.8.2.1 shows the 18 circuits generated for a simple 2 input, 1 segment specification. This shows that negation (the +1 in the affine function box) is toggled during every advance, the segment function is advanced after that (V, then $V^{\dagger}$, then NOT). Finally the input sampling is advanced – first only a, then only b, and then a and b. Note that sampling zero inputs is not a valid configuration.

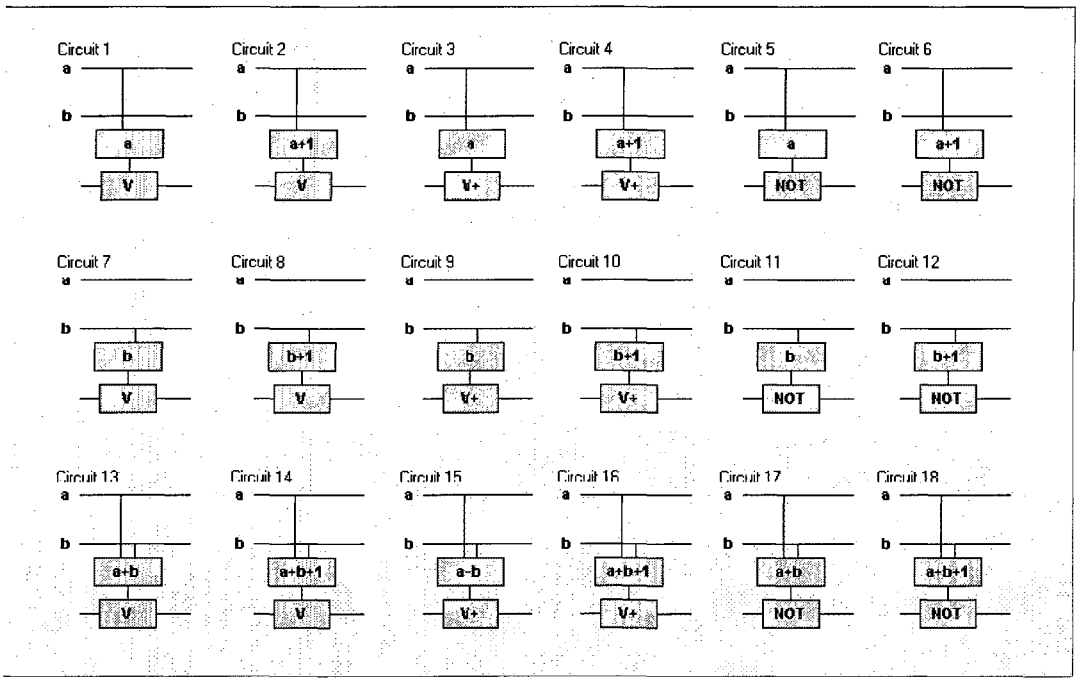


Figure 7.8.2.1: CircuitSearch generated 18 circuits for a simple 2 input, 1 segment specification.

This process scales up to any number of inputs. When additional segments are added, SegmentGenerators can be cascaded. When one reaches the end, the later ones are copied and advanced one after the other. In this way, the generator has the ability to skip large areas of the search space that would result in circuits that are functionally identical (e.g. segments are identical but in a different order).

Description of Main Methods:

IterativeDeepeningCircuitGenerator: This is the constructor that is Responsible for creating the segment generators.

GenerateCircuit: Responsible for creating and configuring new circuit.

Advance: At each *advance* of the generator, the configuration is tested for validity. If the configuration has no inputs it is invalid. If the functional gate has two flags, it is invalid, as there is no mapping for that. Finally, the first segment of a circuit can require that the functional gate is V, anything else is rejected.

Pseudo code:

The algorithm works by dividing each segment up into arrays. Each element of each array represents some configuration of the circuit. Each input line represents one element each, negation of the affine function is one element, and the functional gate (V, $V^{\dagger}$ and NOT) is represented by two elements. Two elements are required because it must represent 3 values. The fourth state (when both elements are flagged) is invalid.

Flags indicate which elements are switched on. Initially there is only one flag, and it steps through all elements. Once that reaches the end, a second flag is introduced, and they start from the left. The second flag is stepped through, and then they start from the second element. This process continues until there are as many flags as there are elements and there are no more circuit configurations to find.

At each *advance* of the generator, the configuration is tested for validity. If the configuration has no inputs it is invalid. If the functional gate has two flags, it is invalid, as there is no mapping for that. Finally, the first segment of a circuit can

require that the functional gate is V, anything else is rejected. For multi-segment circuits, the process is identical. The arrays are concatenated so that flags can iterate over all elements in all segments. The Figure 7.8.2.2 shows the process, with arrays, flags, generated circuits, and reasons for invalidation.

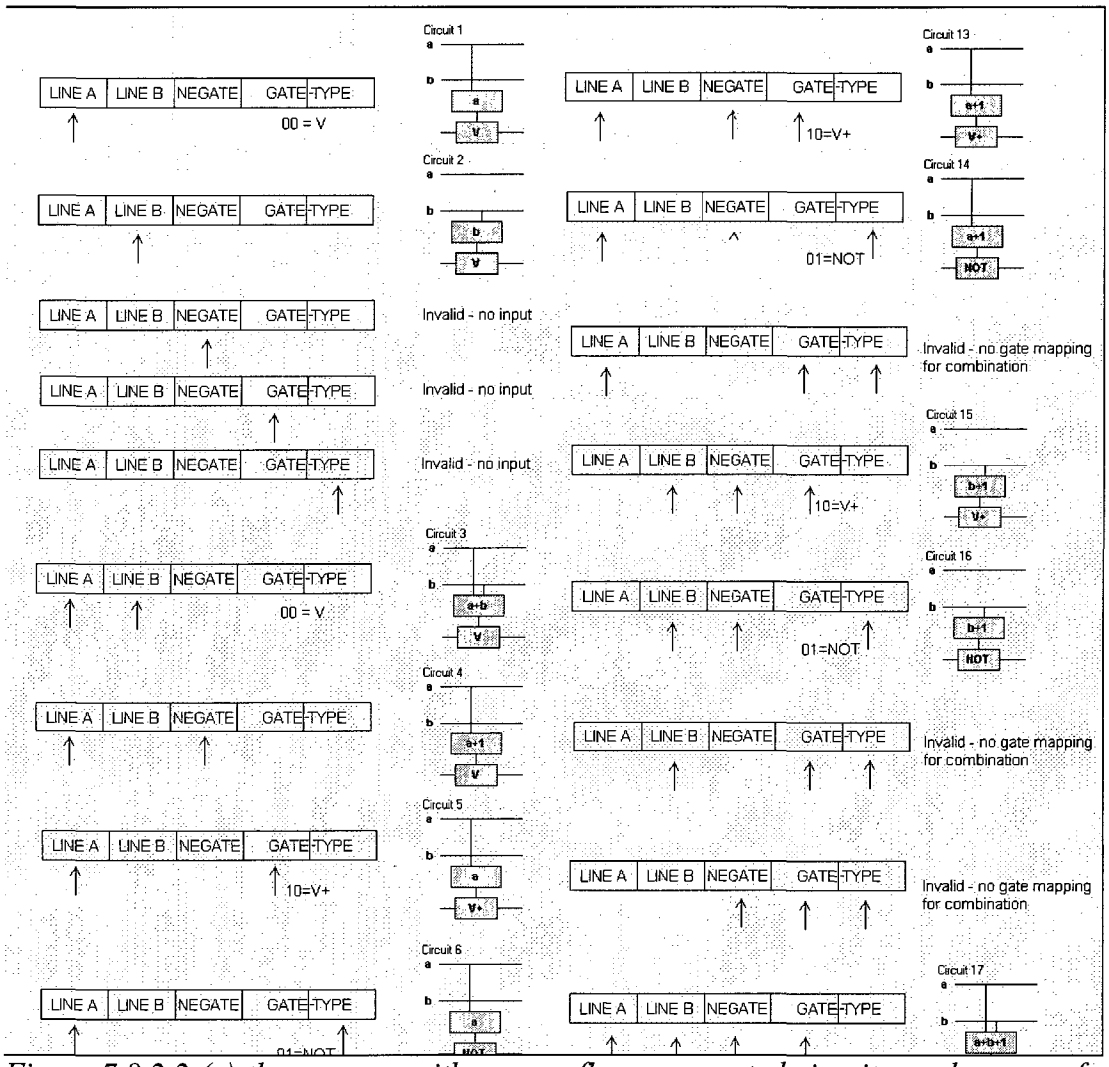


Figure 7.8.2.2:(a) the process, with arrays, flags, generated circuits, and reasons for invalidation in CircuitSearch Program.

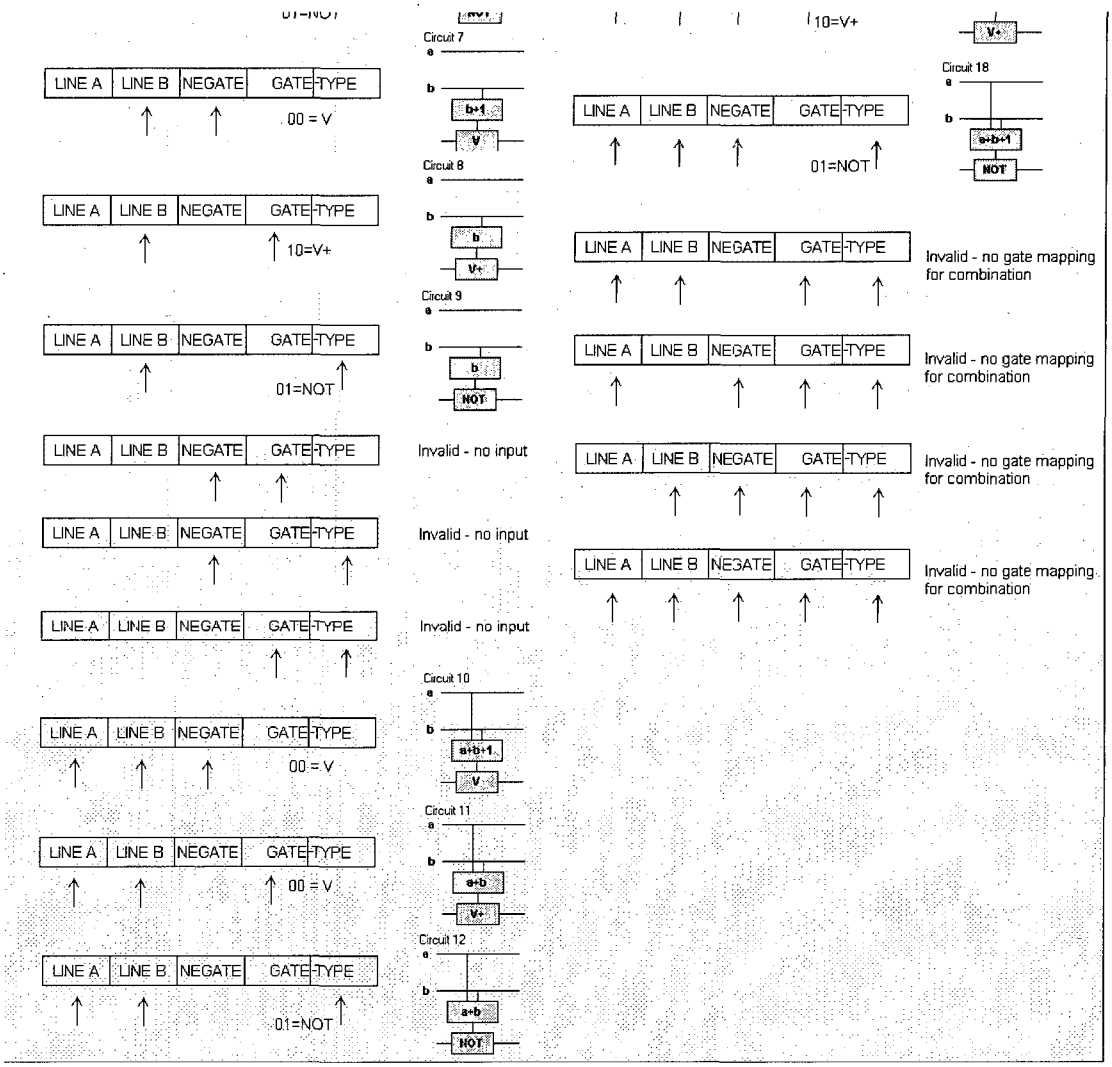


Figure 7.8.2.3: (b) the process, with arrays, flags, generated circuits, and reasons for invalidation in CircuitSearch Program.

7.8.3. How the Iterative Deepening Algorithm Works?

This is similar to exhaustive search, but it varies in the order or circuit generation. It enumerates all circuits at a particular level of complexity before advancing to the next level. Even within a short time you can survey many useful, but perhaps simple, circuits. Because of the way the search space is enumerated, it is unable to skip the

areas mentioned above that exhaustive can. This means that it will, if left to run to completion, generate more circuits, some of which will be functionally identical.

The algorithm works by dividing each segment up into arrays. Each element of each array represents some configuration of the circuit. Each input line represents one element each, negation of the affine function is one element, and the functional gate (V, $V^{\dagger}$ and NOT) is represented by two elements. Two elements are required because it must represent 3 values. The fourth state (when both elements are flagged) is invalid.

Flags indicate which elements are switched on. Initially there is only one flag, and it steps through all elements. Once that reaches the end, a second flag is introduced, and they start from the left. The second flag is stepped through, and then they start from the second element. This process continues until there are as many flags as there are elements and there are no more circuit configurations to find.

At each *advance* of the generator, the configuration is tested for validity. If the configuration has no inputs it is invalid. If the functional gate has two flags, it is invalid, as there is no mapping for that. Finally, the first segment of a circuit can require that the functional gate is V, anything else is rejected.

For multi-segment circuits, the process is identical. The arrays are concatenated so that flags can iterate over all elements in all segments.

7.8.4. Searching.

7.8.4.1. What happens when the Search button is hit?

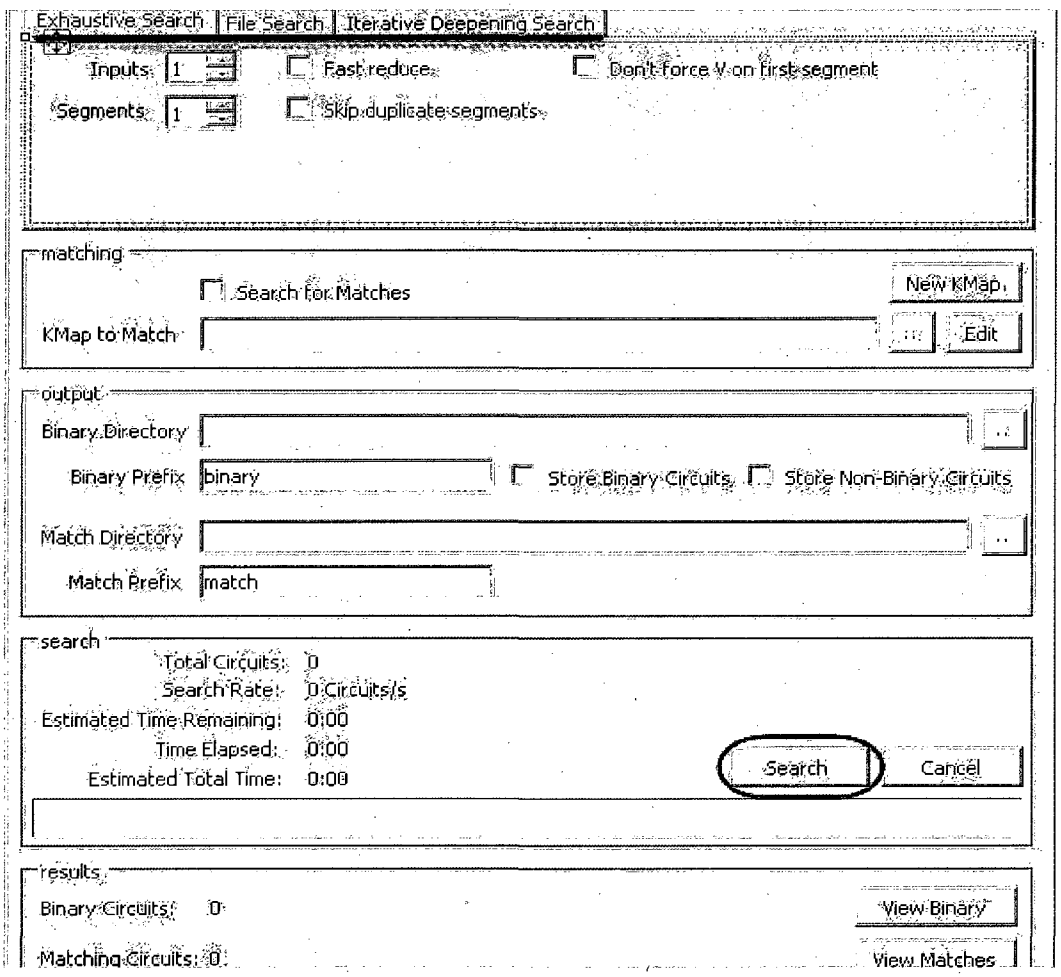


Figure 7.8.4.1.1: Browser of CircuitSearch Program.

This corresponds to the CircuitSearchFrontEnd.Form1.ActionButton_Click() method. This method prepares for the search, locks the user interface from making changes to the configuration during the search, and then kicks off the background worker called

'**searcher**'. The application must be multithreaded to avoid blocking the user interface, as searching is very processor intensive.

The main work loop is in **searcher_DoWork**, where the appropriate generator is created and configured.

For all types of search ExecuteSearch method is implemented:

ExecuteSearch(worker, m_Generator);

Which loops over the GenerateCircuit() and Advance() for the generator, ExecuteSearch tests and writes circuits to file, one file is to record matching circuits, and the other is to record binary circuits.

The search can be paused and saved before completion. A saved search can be loaded and resumed.

7.8.4.2. Abstract algorithm:

- generate all circuits that have given number of segments and given number of inputs
- loop on each circuit
  call "calculateKmaps"
  if current Kmap can not be simplified to a binary Kmap
  then ignore it
  else store the circuit into array of circuits.

7.9. Comparison of Search Techniques and discussion of results.

The two search techniques perform a similar job, but have different strengths. Exhaustive search tends to be faster, as it can skip useless circuits that iterative deepening can't, and does not generate invalid configurations requiring validation. Iterative deepening, however, can quickly discover useful circuits of moderate complexity without bogging down looking at circuits that have some complex segments, and others simple. Both will take vast amounts of time and drive space to complete for non-trivial circuits, but the iterative deepening search will give a broader scan in a reasonable amount of time.

A sample run on a 4 input, 4 segment circuit produced the following timings:

Exhaustive: 2 minutes, 16 seconds. 21870000 binary circuits found.

Iterative: 4 minutes, 24 seconds. 20441521 binary circuits found.

The output circuits file was above 700MB both cases.

Example 7.9.1:

For single input a, there will be 2 => 2^2 = 4 possible functions to check(in KMap, single cell) (so, we should take away the constraint of CircuitSearch that first segment will start with only V, it will search for all possible V, $V^{\dagger}$, NOT, that means for three option, only NOT gate will generate binary results and will be counted as output function). For 2 input (a, b) single output functions (2 × 1), there will be 2 ×2 => 2^4 =

$4 \times 4 = 16$ possible functions . Same for 3 input (a, b, c) single output (3×1) functions there will be $2 \times 2 \times 2 \Rightarrow 2^8 = 16 \times 16 = 16^2 = 256$ possible functions.

For 4 input single output (4×1) functions there will be $2 \times 2 \times 2 \times 2 \Rightarrow 2^{16} = 256 \times 256 = 256^2 = 65536$ possible functions. For 5 input single output (5 X 1) functions, there will be $2 \times 2 \times 2 \times 2 \times 2 \Rightarrow 2^{32} = 65536 \times 65536 = 65536^2 = 42,94,967,296$ possible functions. Means astronomical.

We know for 3 * 3 reversible functions we that have a total of 8!(=40,320) functions, but we examined the target bit output, means single output functions which generates relatively less circuits as it is seen from the above example. In our Software, the maximum number of inputs is set to 30, due to the use of 32 bit integers to represent the input masks. Circuits with this number of inputs would be impractical to search using these techniques anyway – as the search space would be astronomical. Here we developed Libraries for Quantum Circuit restricting our primitive gates to $V, V^{\dagger}$ and NOT. Practically, it depends on technology, but in present most viable quantum circuits are basically with these three gates (V, $V^{\dagger}$, NOT). However, we use affine function to realize our Quantum circuits which gives us many advantages to build practical Quantum Circuits. The advantages are that it is simple and it is technology independent. It is also similar to its classical counter part. In our program, the synthesis of algorithm based on the principle that in each step algorithm perform a

choice of a gate (our definedV, $V^{\dagger}$,NOT) to be added with any possible affine function of inputs.

Example 7.9.2:

Tofolli Gate Search Matching:

In Exhaustive Search: It needs 5 segments, total circuits searched is 43,563,744, 404,278 Circuits searched and total time required 60 seconds. Total binary Circuits found for that 1,758,456 circuits and total matched Circuits 4,584.

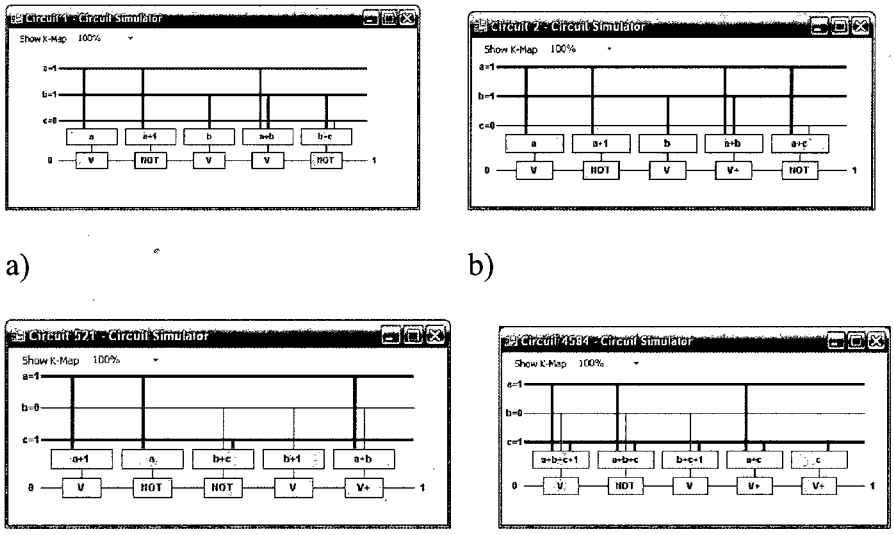


Figure 7.9.2.1: Examples of Circuit simulator interface.

Iterative Search: Same for 5 segments, total circuit searched unknown. Search rate 1,080 Circuits/s and total time needed is 3 minutes 52 seconds, almost 4 times more time than exhaustive search. Total binary circuits found is 801,444 means search space is much less than exhaustive search and matching circuit found is 2,952 which is 50%

of the circuits found in Exhaustive search. Here for iterative deepening, we used fast reduce option.

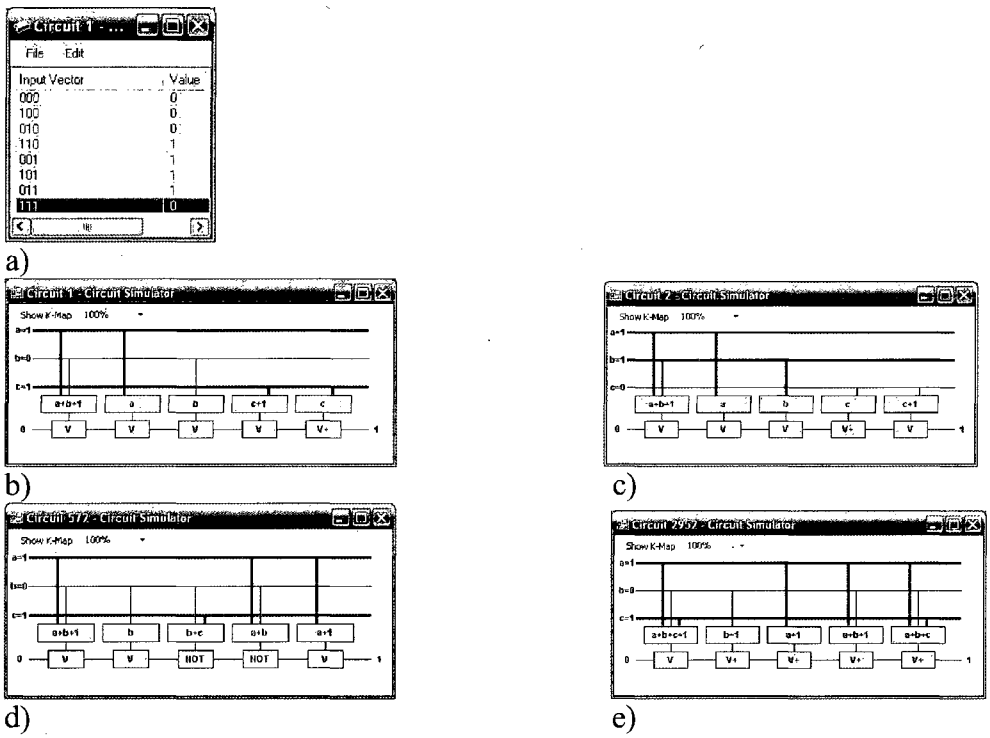


Figure 7.9.2.2: More corcuits found automatically by CircuitSearch.

Test	No. of inputs	No. of segments	Total Circuits tested	Total Circuits found	Search time in sec	Search rate Circuits/ sec
1	1	1	12	6	0.203	59.1133
2	1	2	78	30	0.015	5200
3	1	3	364	115	0.006	-
4	1	4	1365	387	0.016	85,312.5
5	1	5	4368	1,148	0.062	70,451.6
6	2	1	24	10	0.0013	-
7	2	2	300	76	0.0024	-
8	2	3	2,600	461	0.031	83,871
9	2	4	17,550	2,461	0.25	70,200
10	2	5	98,280	11,782	0.985	99,776.6
11	2	6	475,020	51,512	4.844	98,063.6
12	2	7	2,035,800	207,184	21.984	92,603.7
13	2	8	7,888,725	772,235	95.594	82,523.2
14	3	1	48	18	0.031	1,548.39
15	3	2	1,176	216	0.016	73,500
16	3	3	19,600	2,097	0.188	104,255
17	3	4	249,900	18,025	1.953	127,957

Table 7.9.2.1: Complexity evaluation for some results of CircuitSearch.

Test	No. of inputs	No. of segments	Total Circuits tested	Total Binary Circuits found	Matched Circuits	Search rate Circuits/ sec
1	3	2	588	28	-	294
2	3	3	24,696	1,288	-	12,348
3	3	4 (Peres)	1,037,232	48,566	P-612	518,616
4	3	5(Toffoli)	43,563,744	1,758,456	T-4,584	85,312.5
5	2	2	108	12	CN-1	54
6	2	3	1,944	216	CN-12	972
7	2	4	34,992	3,486	CN-220	17,496
8	2	5	629,856	56,040	CN-3,736	314,928
9	2	6	11,337,408	910,356	-	246,414
10	2	7	204,073,344	14,958,456	-	234,477

Table 7.9.2.2: Generating matched circuits CircuitSearch Program using exhaustive search.

Test	No. of inputs	No. of segments	Total Circuits tested	Total Binary Circuits found	Search rate Circuits/ sec
1	3	2,mx	Unknown	15	189
2	3	2,3	Unknown	4	26
3	3	2,5	Unknown	10	128
4	3	2,7	Unknown	14	185
5	3	2,9	Unknown	15	189
6	3	2,mx	Unknown	700	7,500
7	3	3,4	Unknown	18	84
8	3	3,5	Unknown	45	494
9	3	3,6	Unknown	159	1,535

Table 7.9.2.3: Generating matched circuits CircuitSearch Program using iterative deepening search.

Minimum cost: Approximation: all V, $V^{\dagger}$, NOT gates cost same = 1. Gates (V, $V^{\dagger}$, NOT) with single EXOR cost = 2, EXOR of 2 (Like a EXOR b with any gates V, $V^{\dagger}$, NOT) cost = 3 and so on.

When we analyzed the results of the CircuitSearch program (Tables 7.9.2.1 and 7.9.2.2) we found the following:

1. When the circuits become larger, the higher proportions of them are not permutative, thus the method wastes a lot of time to find nothing useful. We noted the patterns (explained earlier) in these functions.

2. There are very many circuits for the same function. When the numbers of variables grows, the same functionality is obtained in extremely many circuits.

Again this means that there is no need to use this software for large functions. These patterns for small functions that we found are the fundaments of our methods.

3. Analyzing our designs found by the software we found however interesting properties and patterns that are independent on the numbers of variables (see for instance the interval functions).

4. We found that a very interesting property, Property 7.9.1, is true.

Property 7.9.1:

1. Given is a quantum array built with only CV, $CV^{\dagger}$, NOT and CNOT gates
2. We replace all CV and $CV^{\dagger}$ gates with CV and $CV^{\dagger}$ gates in all possible ways
3. We remove or add any number of NOT and CNOT gates in arbitrary way to the structure
4. Then the function realized by this modified array remains permutative.
5. Any other transformation (replacement, addition or removal) leads to a non-permutative circuit.

Based on the above Property 7.9.1, the CircuitSearch program proved to be a very useful prototyping tool as it allowed to find a general property which was not known

earlier. This property allowed me to create a library of inexpensive gates to be used in hierarchical synthesis methods.

7.10. Library based design

7.10.1. Design of library of reversible blocks for single-output functions.

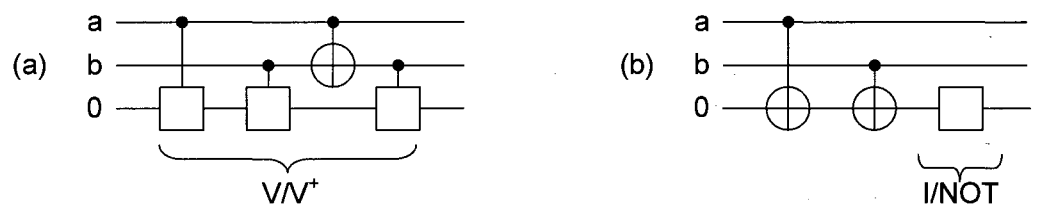


Figure 7.10.1.1: Patterns of the least expensive realizations of functions of 2 variables. (a) Empty block can be any V or V^{+}, (b) empty block can be any of identity or inverter.

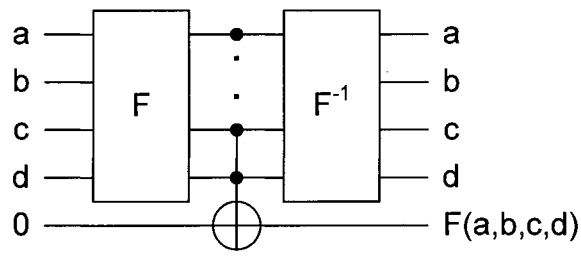


Figure 7.10.1.2: Pattern of all gates in 5×5 library of 4-argument functions. F is an arbitrary 4×4 reversible function.

All single output functions of 2 variables can be implemented in library cells from Figure 7.10.1.1 a,b. The idea of all cells for 2,3 and 4 variables is shown in Figure 7.10.1.2. The classification (NPN classification) of switching functions is an important problem in logic design used for the development of universal logic modules and for cell library binding [Perkowski95, Perkowski01]. The key to such an approach is to determine that two functions are equivalent relative to a permitted set of operations. Here we consider two functions to be matchable if they are equivalent relative to permitted set of operations. Cell library binding, also frequently called technology mapping, is the process of transforming arbitrary free logic circuits where the interconnection of components are the instances of basic elements from a given library. The cell library binding uses some type of classification of switching functions as a tool to match efficiently the library instances with free logic circuits [Perkowski95, Perkowski01]. The principal algebraic classification method [Harison65] considers the following operations, taken individually or collectively: negation (N) of one or more input variables of the functions, permutation (P) of two or more of the input variables of the functions and negation (N) of the output of the functuions. Boolean functions that are equivalent under negation of inputs are N-equivalent, under permutation of inputs are P-equivalent, and under both stated conditions, are NP-equivalent [Harison65]. If the complementation of the output is also considered the Boolean functions are NPN-equivalent. The canonical forms of NP-equivalent Boolean functions are identical. In other words, two Boolean functions are NP-equivalent exactly if they share the same canonical forms, called the representative functions.

It is well known that NP and NPN matching are useful techniques for synthesis of combinational functions with gate libraries.We use the concept of NPN classification to develop our library. It is explained in Figures 7.10.1.3, 7.10.1.4, 7.10.1.5 and 7.10.1.6.

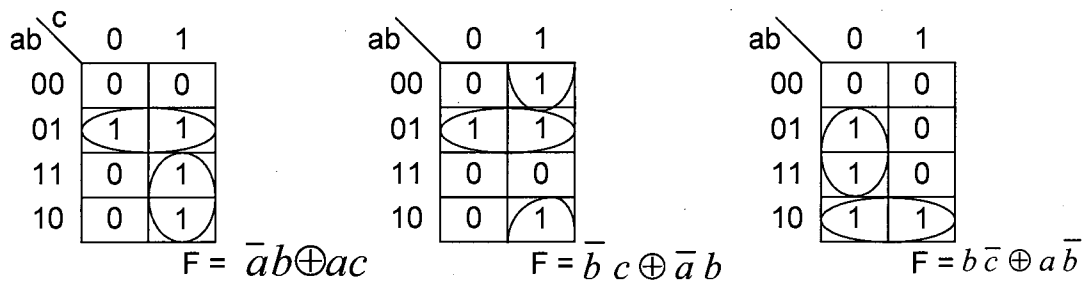


Figure 7.10.1.3: Shows patterns of 3×3 Fredkin-Like gates. They are all NPN compatible.

We can also say that they are all in the same category of classification. Each of them can be a representative function of this category.

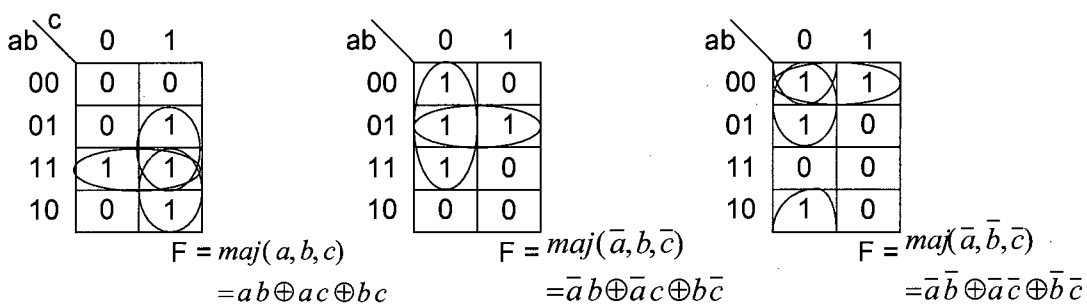


Figure 7.10.1.4: This figure illustrates patterns of majority function of 3 variables with 3 polarities, abc, $\bar{a}\,b\,\bar{c}$ and $\bar{a}\,\bar{b}\,\bar{c}$ resepectively.

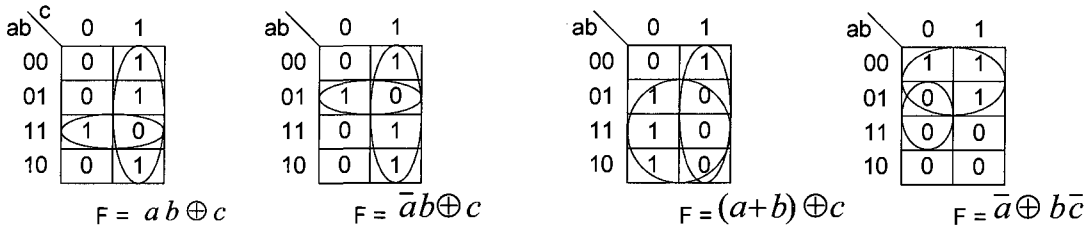


Figure 7.10.1.5: Shows pattern of Toffoli-Like 3×3 gates. They are all in the same NPN class.

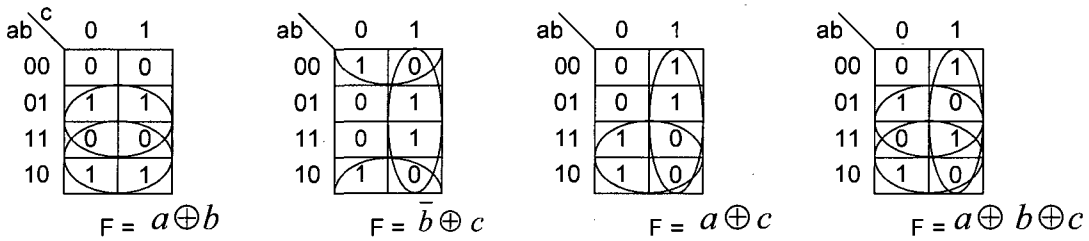


Figure 7.10.1.6: Patterns of affine (Feynman-Like) 3×3 gates. Three first are in the same NPN class.

The library is to be used in conjunction with the decomposition algorithm. Therefore all cells (reversible blocks) from the library are reversible and return the original inputs a, b, c to their outputs. All additional ancilla bits in the cells are returned to $|0\rangle$.

This way, all the cells from the library can be stacked as tiles in a reversible layout for larger functions. This is shown in Figure 7.10.1.7 below. This way, every large function is decomposed to cells from our library.

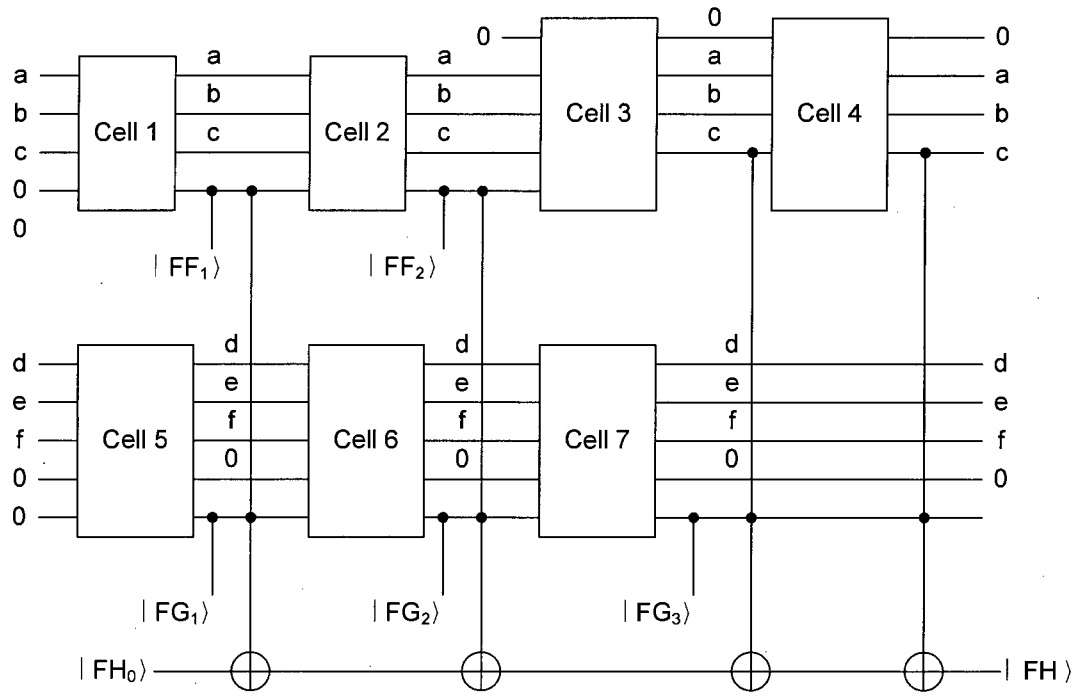


Figure 7.10.1.7: A general method to realize a single-output function of many variables $|FH\rangle$ using cells of 3-variable library.

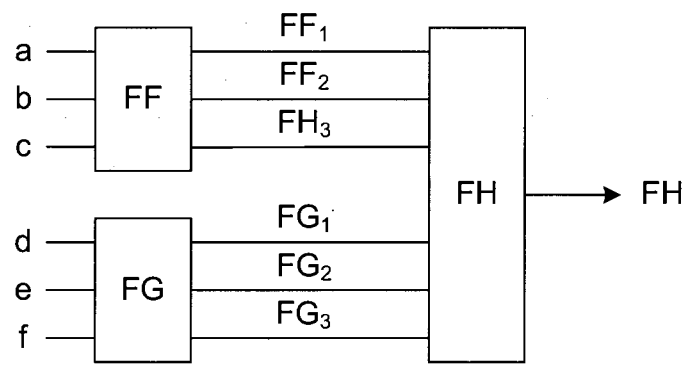


Figure 7.10.3.8: The original decomposition of non-reversible function FH to be next realized as a reversible function using our library of reversible cells. It is just an example of decomposition.

The reversible circuit from Figure 7.10.1.7 is created from a non-reversible decomposition presented in Figure 7.10.1.8.

Let us explain now the method how all the cells of the reversible library were created. It was first necessary to find a representative function from each NPN classification category of 3-variable functions. We selected always the least expensive function (in terms of the number of inverters).

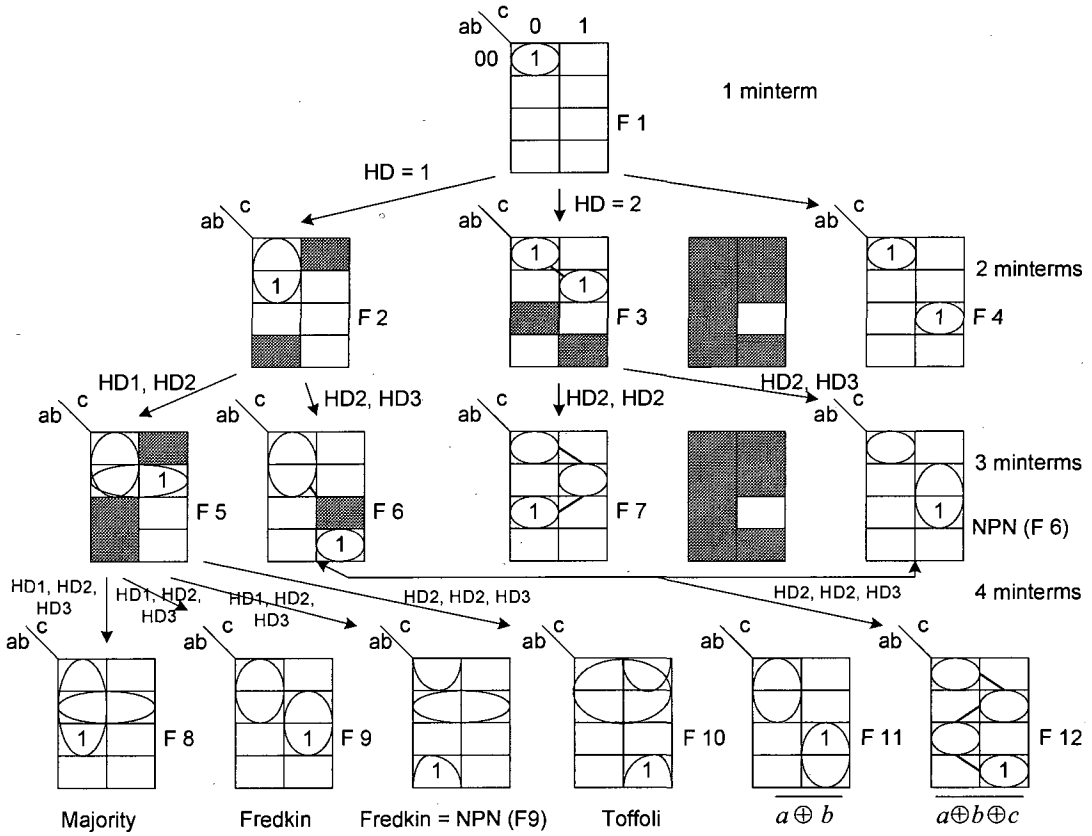


Figure 7.10.1.9: Creation of NPN equivalent functions of three variables.

Figure 7.10.1.9 shows the systematic tree search method to find the representative function for each NPN class. The levels of the tree correspond to functions with the

increased number of true minterms. We know that each minterm can be represented by a product $a^i b^j c^k$, i, j, k $\in$ {0, 1} so each minterm is a representative of the NPN class $a^i b^j c^k$. We select minterm $\bar{a}\,\bar{b}\,\bar{c} = a^0 b^0 c^0$ as the representative function of the class of single-minterm functions $a^i b^j c^k$.

The KMap corresponding to this function is on top of the tree of all representatives. It is denoted by F1. Now we have to add another minterm in all possible ways to create representatives of all 2-minterm (and $2^3 - 2 = 8 - 2 = 6$ minterm) classes. Second row of the tree has all such functions. Function F2 is representative of all functions in which the second added minterm (shown as "1" in KMap for F2) is in Hamming Distance 1 (shown as arrow with HD = 1 that points from F1 to F2). Other minterms in HD = 1 (HD1) from $m_0 = \bar{a}\,\bar{b}\,\bar{c}$ are shaded as not to be considered further on this level of the tree. By adding the second minterm with HD2 the representative function F3 is created. All minterms not to be considered for addition in this level of the tree are shaded (KMap right to F3). Thus the only remaining representative of the 2-minterm class is F4.

Using the same method the next level of the tree is created. F5 has distances HD1 and HD2 to true minterms of F2 from the previous level of the tree. For didactic reasons function NPN(F6) is shown at right but other representatives of the same NPN class are not given. This way all NPN representatives F5, F6 and F7 of functions with 3 and 2^3-3 = 5 minterms were exhaustively found.

Next level of the tree creates functions F8, F9, F10, F11 and F12, all with 4 true minterms – these are all balanced functions that are not literals.

Observe that:

$$F8 = \overline{a}\,\overline{c} + b\,\overline{c} + \overline{a}\,b = \overline{a}\,\overline{c} \oplus b\,\overline{c} \oplus \overline{a}\,b$$
$$= majority(\overline{a}, b, \overline{c}) = NPN(majority(a, b, c))$$
$$= a\,b \oplus a\,c \oplus b\,c = a(b \oplus c) \oplus b\,c$$

Thus function $a(b \oplus c) \oplus b\,c$ can be realized as a NPN class representative.

Observe that: F9 is half non-variable outputs of Fredkin gate (outputs that are not variables).

$$F9 = \overline{a}\,\overline{c} \oplus b\,c = mux(c; \overline{a}, b)$$
$$= NPN(mux(c; \overline{a}, b)) = \overline{a}\,c \oplus a\,b$$

(another NPN(F9) is drawn next to F9).

Observe that variable output of Toffoli gate is:

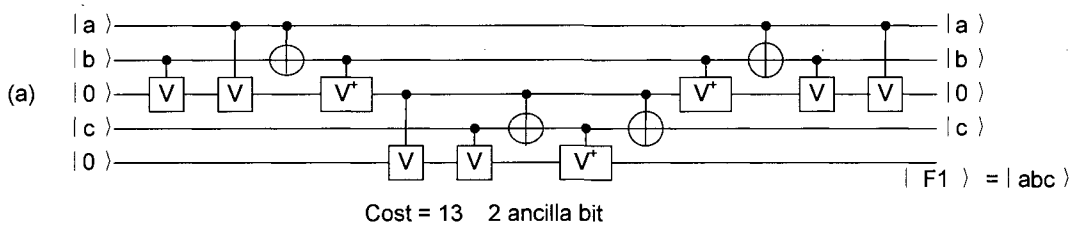
$$F10 = \overline{a} \oplus \overline{b}\,\overline{c} = NPN(a \oplus b\,c)$$

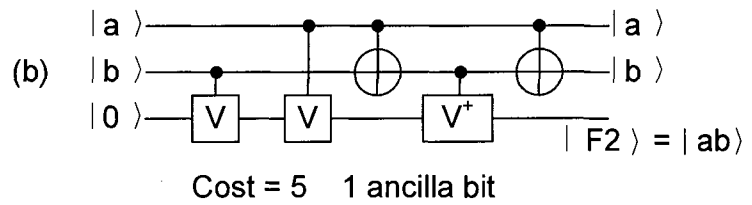
Now that all NPN equivalent classes have been found we will realize a library cell for one representative, the one with the smallest number of inverters. All other functions from this class can be created by 4 methods:

1. permuting inputs
2. negating inputs

3. negating the output

4. replacing V → $V^{\dagger}$ and $V^{\dagger}$ → V, or any combination of these four methods.





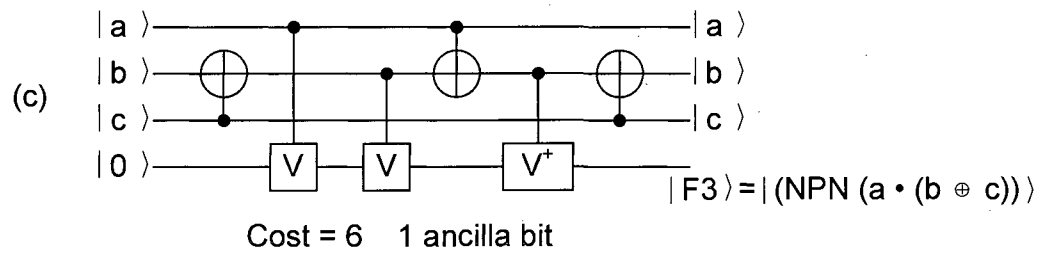


Figure 7.10.1.10: Example realization of library cells for all NPN equivalent functions of three variables. (a) function $|F_1\rangle = |abc\rangle$, *(b) function* $|F_2\rangle = |ab\rangle$, *(c) function* $|F_3\rangle = |a(b \oplus c)\rangle$.

Figure 7.10.1.10a presents the library cell for NPN(F1) class = $|abc\rangle$. It has cost 13 2×2 quantum primitives and 2 ancilla bits. Observe again that a product which is considered an expensive function in classical PLA logic is here one of the most

expensive functions. Figure 7.10.1.10b presents the library cell for NPN(F2) class = $|ab\rangle$. It has cost 5 and 1 ancilla bit. Figure 7.10.1.10c presents the library cell for NPN(F3) = a (b $\oplus$ c). It has cost 6 and one ancilla bit. Figure 7.10.1.11 shows the NPN class representative of NPN ($|F4\rangle$). It realizes function $\overline{a}\,\overline{b}\,\overline{c} + a\,b\,c = \overline{a}\,\overline{c} \oplus b\,\overline{c} \oplus a\,b = \overline{c}\,(\overline{a} \oplus b) \oplus a\,b$. It has cost 10 of 2×2 primintives and additional 4 inverters and one ancilla bit. Another realization of NPN ($|F4\rangle$) is shown in Figure 7.10.1.12. It has cost 8 and one ancilla bit.

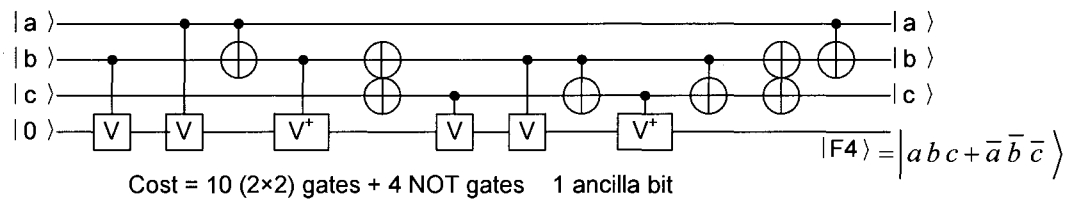


Figure 7.10.1.11: Realization of NPN class of Function $|F4\rangle$.

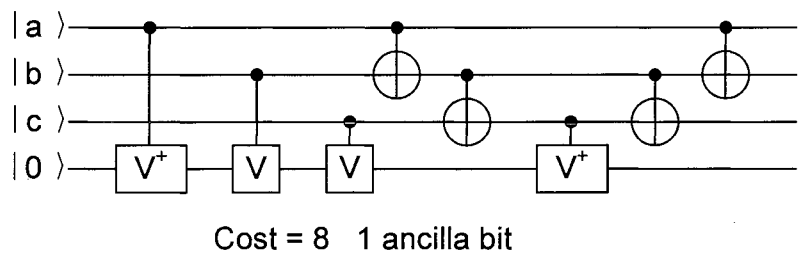


Figure 7.10.1.12: Another realization of NPN ($|F4\rangle$).

(a) $\bar{a} \oplus \bar{a}\,\bar{b}\,c = \bar{a}(1 \oplus \bar{b}\,c) = \bar{a}(b + \bar{c}) \underset{NPN}{\approx} a(b + \bar{c})$

(b)

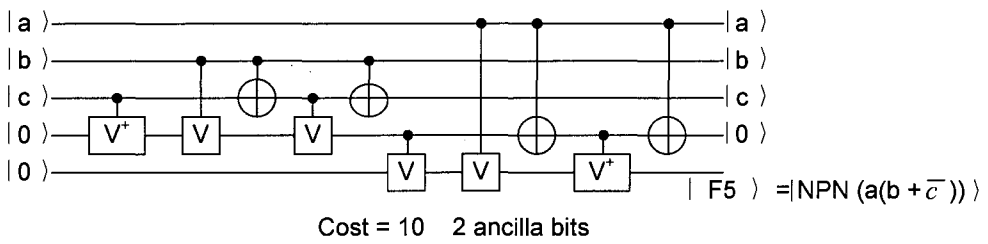
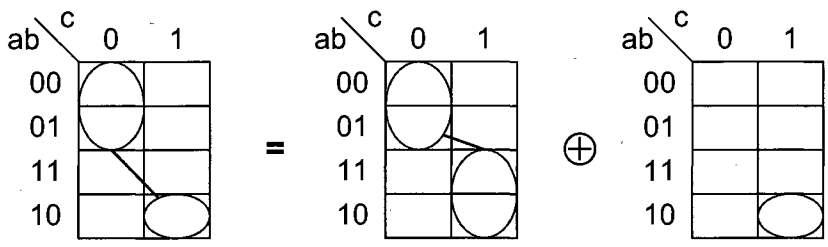


Figure 7.10.1.13: Realization of NPN ($|F5\rangle$).

Figure 7.10.1.13 presents synthesis of NPN($|F5\rangle$). Figure 7.10.1.13a has logic transformation and Figure 7.10.1.13b has the final circuit with cost 10 and 2 ancilla bits. Figure 7.10.1.14a explains using KMaps the decomposition of $|F6\rangle$. Its circuit realization is presented in Figure 7.10.1.14b. This has cost 14 and two ancilla bits. Figure 7.10.1.15a explains using KMaps the decomposition of $|F7\rangle$. Formulas for NPN representative are given in Figure 7.10.1.15b. The final circuit has cost 15 and 2 ancilla bits and is shown in Figure 7.10.1.15c.

(a)



$$|F6\rangle = \bar{a} \oplus c \oplus abc = NPN(a \oplus b \oplus abc)$$

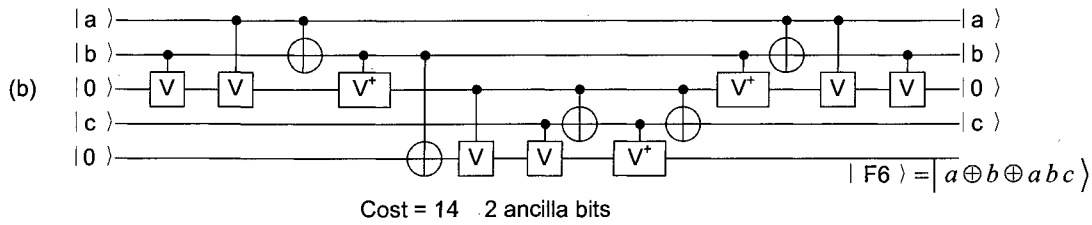
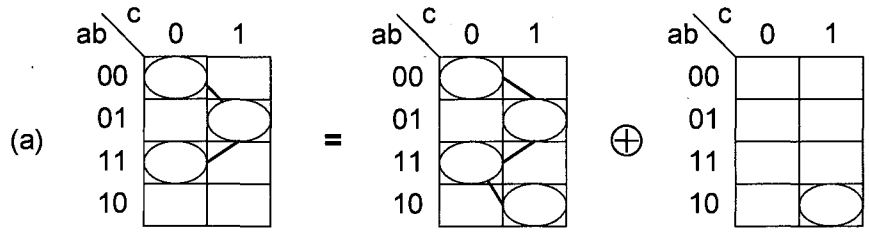


Figure 7.10.1.14: Realization of NPN class of function $|F6\rangle$. (a) Exor decomposition: illustrated using KMaps, (b) the final quantum array.



(b)

$$|F7\rangle = (a \oplus b \oplus \bar{c}) \oplus a\,\bar{b}\,c$$
$$= NPN\,[\,a \oplus \bar{b} \oplus c \oplus a\,\bar{b}\,c\,]$$
$$= NPN\,[\,a \oplus b \oplus c \oplus a\,b\,c\,]$$

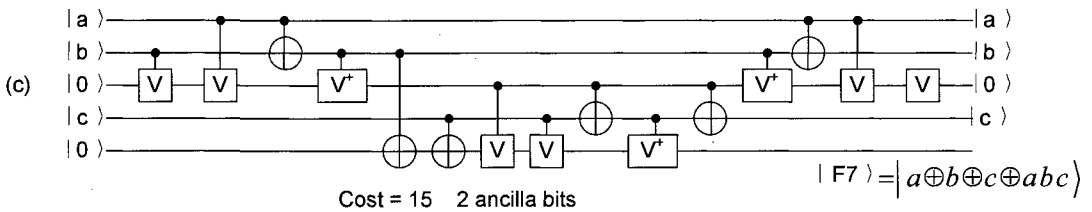


Figure 7.10.1.15: Realization of the library cell for NPN ($|F7\rangle$). (a) Exor decomposition using KMaps, (b) step by step transformations leading to the cheapest representative circuit of NPN ($|F7\rangle$), (c) the final quantum array.

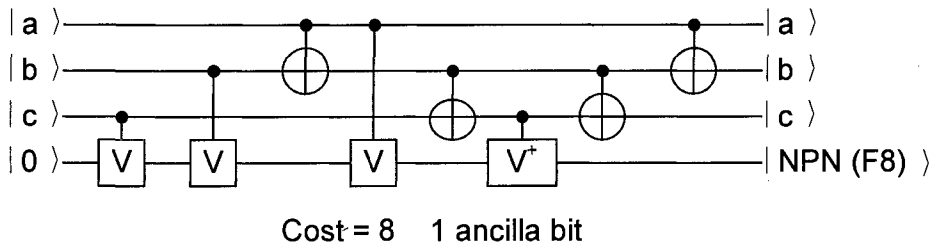
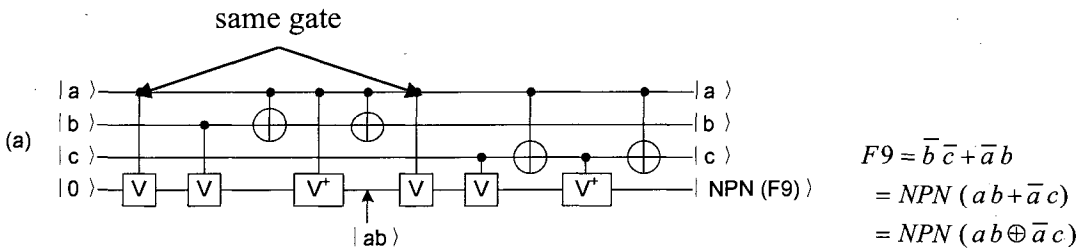


Figure 7.10.1.16: Realization of NPN class function of $|F8\rangle$.

Figure 7.10.1.16 presents the circuit realization of the library cell for NPN($|F8\rangle$). It has cost 8 and one ancilla bit. This cell was found directly by program CircuitSearch and was the base of the whole idea of the least expensive NPN representatives.



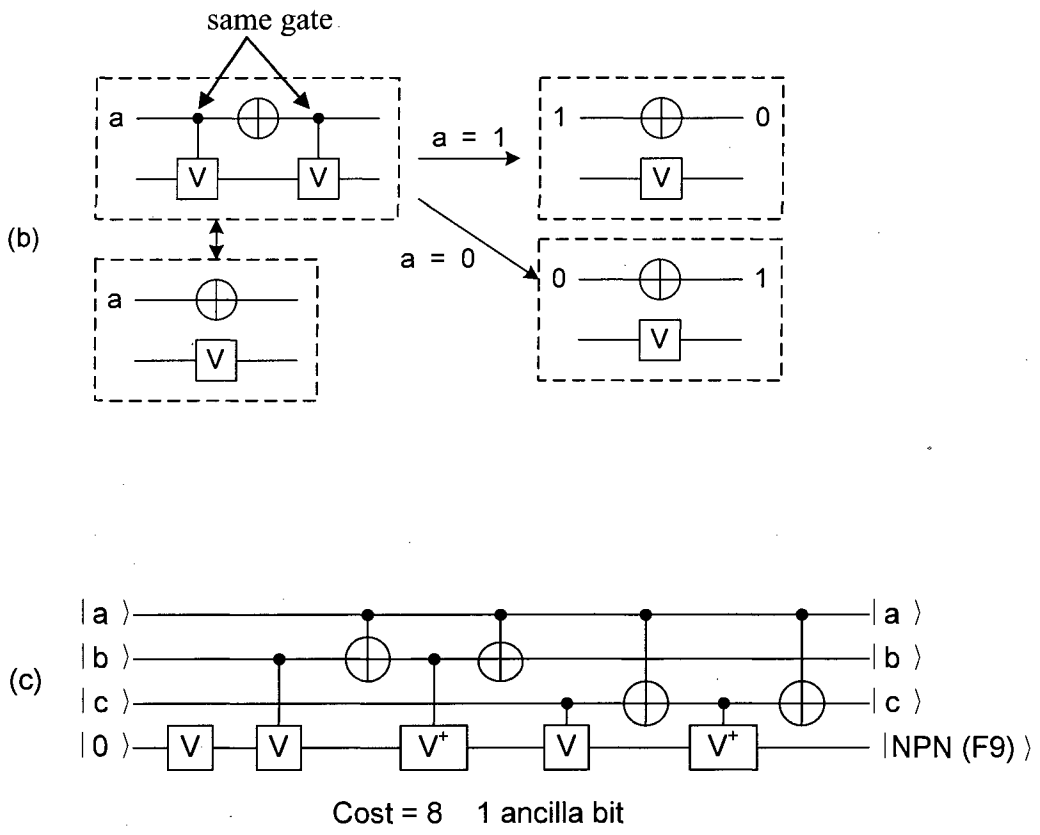


Figure 7.10.1.17: Realization of NPN class function of NPN ($|F9\rangle$). (a) initial, non-optimized circuit, (b) equivalent circuit transformations used in optimization of the circuit from Figure 7.10.1.17a, (c) the final optimized cell for the library.

Figure 7.10.1.17 presents the algorithmic design of NPN($|F9\rangle$). Observe that using the transformations from Figure 7.10.1.17a the circuit from Figure 7.10.1.17b is reduced to the equivalent circuit from Figure 7.10.1.17c which has the cost of 8 2×2 primitives and one single-qubit operator (V) and one ancilla bit. This circuit was also found directly by my program CircuitSearch after its modification to include non-controlled gates V. Figure 7.10.1.18a explains the realization of NPN($|F10\rangle$) = $|ab\oplus c\rangle$ as a library cell. Observe that this realization has the same cost as another realization

obtained from standard Toffoli gate given in left of Figure 7.10.1.18c. But in order to have proper order of outputs in the cell to make it compatible with other cells in the library a SWAP gate is added in Figure 7.10.1.18c which makes our design from Figure 7.10.1.18b the least expensive. It has cost 6 and one ancilla bit. In some designs the standard Toffoli gate can be used as well as the library cell. We do not need to restore the input $|c\rangle$ at the output.

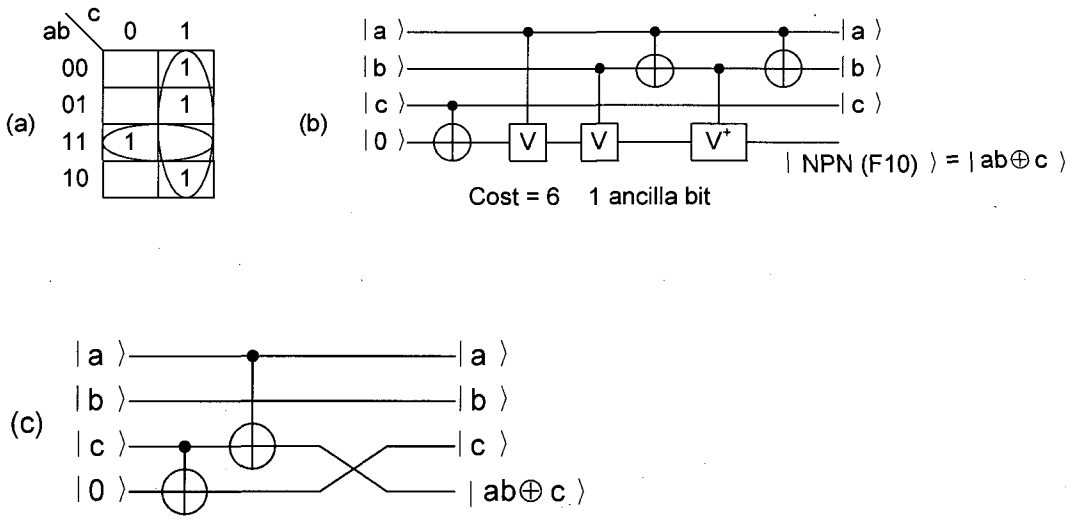


Figure 7.10.1.18: Realization of NPN function in library.

Figure 7.10.1.19a shows the library cell for NPN($|F11\rangle$). It has cost 2 and one ancilla bit. It is one of the cheapest cells, no doubt as it is a pure affine function. Similarly inexpensive is the realization of NPN($|F12\rangle$) from Figure 7.10.1.19b with cost 3 and one ancilla bit. It is also an affine function.

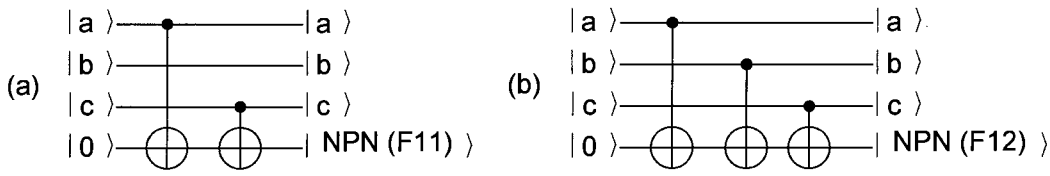


Figure 7.10.1.19: Realization of affine functions NPN(F11) and NPN (F12) as the library cells.

Function NPN #	NPN class	ANC	Cost 2×2 gates	Cost NOT gates	# of minterm	comments	Even/odd
F1	abc	2	13	0	1,7	Most expensive	odd
F2	ab	1	5	0	2,6		even
F3	a (b ⊕ c)	1	6	0	2,6		even
F4	abc + $\bar{a}\bar{b}\bar{c}$	1	10	4	2,6		even
F5	a (b + $\bar{c}$)	2	10	0	3,5		odd
F6	a ⊕ b ⊕ abc	2	14	0	3,5	Most expensive	odd
F7	$a \oplus b \oplus c \oplus a\,b\,c$	2	15	0	3,5	Most expensive	odd
F8	ab + ac + bc	1	8	0	4		even,balanced
F9	ab + $\bar{a}$ c	1	8	1	4		even,balanced
F10	ab ⊕ c	1	6	0	4		even,balanced
F11	a ⊕ c	1	2	0	4	least expensive	even,balanced
F12	a ⊕ b ⊕ c	1	3	0	4	least expensive	even,balanced

Table 7.10.1.1: Costs of gates (cells) in our library (in terms of the number of ancilla bits ANC and the number of 2×2 quantum primitives, as well as the number of inverters.

Table 7.10.1.1 compares the library gates. It is interesting that it confirms numerically our intuition that odd functions are more expensive that even functions and that balanced functions are less expensive on the average then the non-balanced functions. These observations are used in our algorithms from this chapter and will be further used in next chapters.

7.11 New Methodology for Synthesis of Quantum Circuits

7.11.1. General Recursive Decomposition of arbitrary non-reversible Boolean functions to hierarchical quantum circuits with ancilla qubits.

Based on the concepts of affine gates and the results presented in previous sections of this chapter I created a new general methodology to synthesize quantum circuits with greatly reduced quantum costs.

This recursive methodology can be summarized as follows:

Algorithm Decompose

1. If the function F has four or less variables, select the gate from the library of gates presented in section X.
2. If the function F can be represented as a single affine gate AG (of any type) then realize function F according to the synthesis algorithm of gate AG.
3. If F can be represented as F = EXOR_SUM AG_i, i.e. as an EXOR of affine gates, then realize EXOR of recursively realized gates AG_i, adding mirror gates $AG^{-1}{}_i$, if necessary.
4. If F can be represented as F = EXOR_SUM G_i, i.e. as an EXOR of functions G_i of smaller number of variables each, or of simple functions, then realize EXOR of recursively realized gates G_i, adding mirror gates $G^{-1}{}_i$, if necessary.

5. If one affine gate AGi of any type has high correlation with F (i.e. small Hamming Distance of F and AGi) then realize $F = AG_i \oplus F_{rem}$ where F_{rem}, the remainder function, is next realized recursively.
6. In all other cases, apply Ashenhurst/Curtis decomposition, bi-decomposition, or any other type of standard classical combinational logic decomposition of function F, leading to new subfunctions F_i, to be realized recursively by Algorithm Decompose. For each gate (circuit) that realizes F_i mirror gates are created, if necessary.

Below I will present components of this new methodology and I will illustrate them with examples.

7.11.2. Ashenhurst-Curtis Decomposition

It is well-known that the best and most general logic synthesis methods for classical combinational circuits such as Ashenhurst-Curtis decomposition are global and do not depend on gates that are used in the decomposition. These methods should then be also applied to reversible circuits. Example of Ashenhurst-Curtis decomposition is shown in Figure 7.11.2.1.

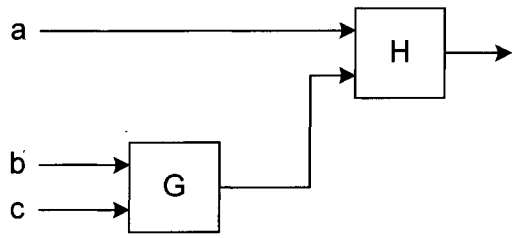


Figure 7.11.2.1: Symbolic representation of Ashenhurst Decomposition. The original function F with inputs a, b, c is decomposed to two subfunctions, G and H. Function G is called the predecessor block and function H is called the successor block. There is only one signal connecting blocks G and H.

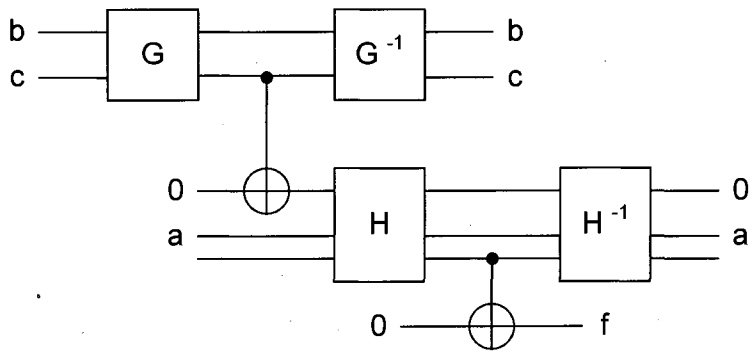


Figure 7.11.2.2: Realization of Ashenhurst decomposition from Figure 7.11.2.1 trasformed to a reversible circuit. As can be seen, there are blocks G^{-1} and H^{-1} added to the circuit. Block G^{-1} is a mirror circuit (inverse) of G and block H^{-1} is an inverse circuit (mirror circuit) of block H. This way, the entire circuit has all its primary inputs restored at its outputs, as necessary in Grover oracles. Some ancilla bits are added, initialized to constant and, with exception of output f, restored to these constants. Here G is reversible, so no ancilla bit was added to G. In general an ancilla bit should be added.

The decomposition method illustrated in Figure 7.11.2.2. is applicable to any types of specifications of blocks and functions. It does also not determine how the gates inside blocks G and H are realized. Observe that this method decomposes a function with many inputs to few blocks of smaller size to which other synthesis methods can be applied. This way, after Ashenhurst-Curtis decomposition applied to a many-input

function, the reversible synthesis methods proposed here which are not applicable to very large functions can be successively applied. Moreover, when recursive decompositions create finally small enough functions, the best realizations of these functions can be found in the library of gates. This method creates one ancilla bit for each block pair G/G^{-1} and H/H^{-1}. All except of the last one, these bits terminate with constants which allows to fold more signals to the same qubits. Also, all primary inputs of function F are restored, which can be used in several synthesis methods for circuits in which function F was only one of several functions on arguments a, b, c.

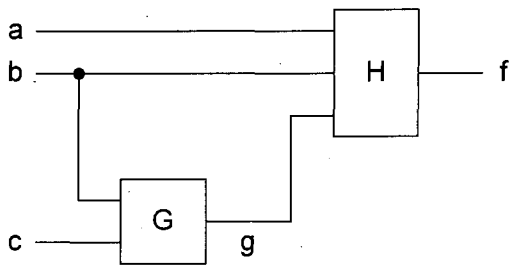


Figure 7.11.2.3: Ashenhurst Decomposition with non-disjoint sets of bound and free variables. Free variables are {a, b} and bound variables are {b, c}. This means that variable c, called the shaded variable is used in both free and bound sets of input variables to blocks H and G, respectively.

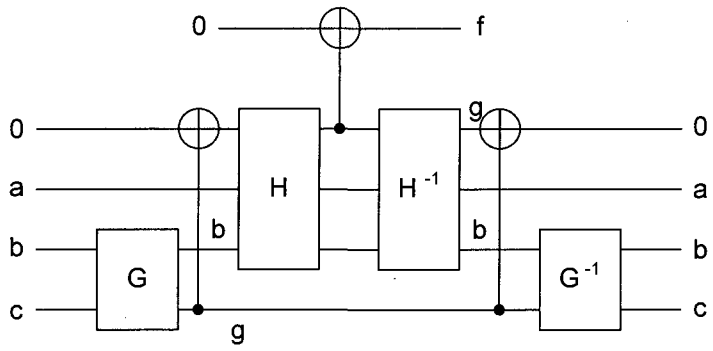


Figure 7.11.2.4: The realization of circuit from Figure 7.11.2.3 in a reversible cascade with reversible blocks G and H and their mirror blocks. Observe that variable b is a go-through variable in blocks G and G^{-1} . Observe also two ancilla bits added, one for the output f and one for intermediate signal g. This ancilla bit is initialized and terminated with constants zero. This figure explains the most general pattern of applying recursively the Ashenhurst-Curtis decomposition to arbitrary (multi-input, multi-output) Boolean function to be realized as a quantum oracle with quantum gates.

Reversible circuits similar to those from Figures 7.11.2.2 and 7.11.2.4 can be designed for any numbers of bits in free and bound sets. The presented here general method works regardless of sizes of sets of shared variables in these sets, possibly empty sets. Bigger examples analyzed by me show also that even if the decomposition is used recursively to every block G or H, again and again, the above decomposition method will still work. In the worst case it may create too many ancilla bits, but their number is usually smaller than using other synthesis methods.

We are not interested in this section 7.11 in the details of decomposition algorithms, many such particular algorithms exist and can be used as a pre-processing step in

reversible logic synthesis methods presented in this thesis. Please understand that each signal a, b, c etc above may represent any number of binary wires.

In contrast to the algorithms from literature [Maslov05] the algorithm from section 7.10.1 can transform a non-reversible function to a reversible one during the synthesis. The presented algorithm solves a non-reversible function by adding ancilla qubits to the input and output to make the entire function reversible. The algorithm can be used to create good solutions for these problems if proper assignments are made to the ancilla qubits, however finding the proper assignment is in general not a trivial exercise.

7.11.3. Using symmetry and regularity to select "simple gates" for generalized Decomposition

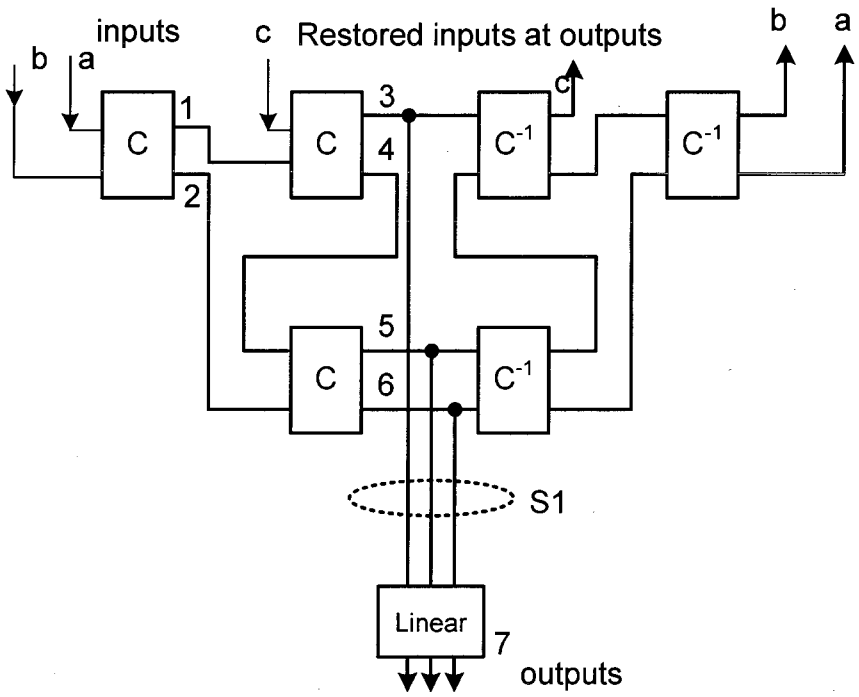


Figure 7.11.3.1: Reversible Net structure to generate all multi-output symmetric functions of variables a, b, c. S1 is the net of symmetric indices from which all symmetric functions can be created. Block C^{-1} is the inverse (mirror) of block C.

In step 4 of Algorithm Decompose is section 7.11.1 we referred to simple functions. This definition is not precise and it may depend on algorithm's implementation. For instance, regularly realized and symmetric functions may be regarded as simple. For instance the structure called Reversible Net in Figure 7.11.3.1 realizes arbitrary multi-output symmetric function in a regular pattern of connections and restores all inputs to their orginal values at the circuit's outputs.

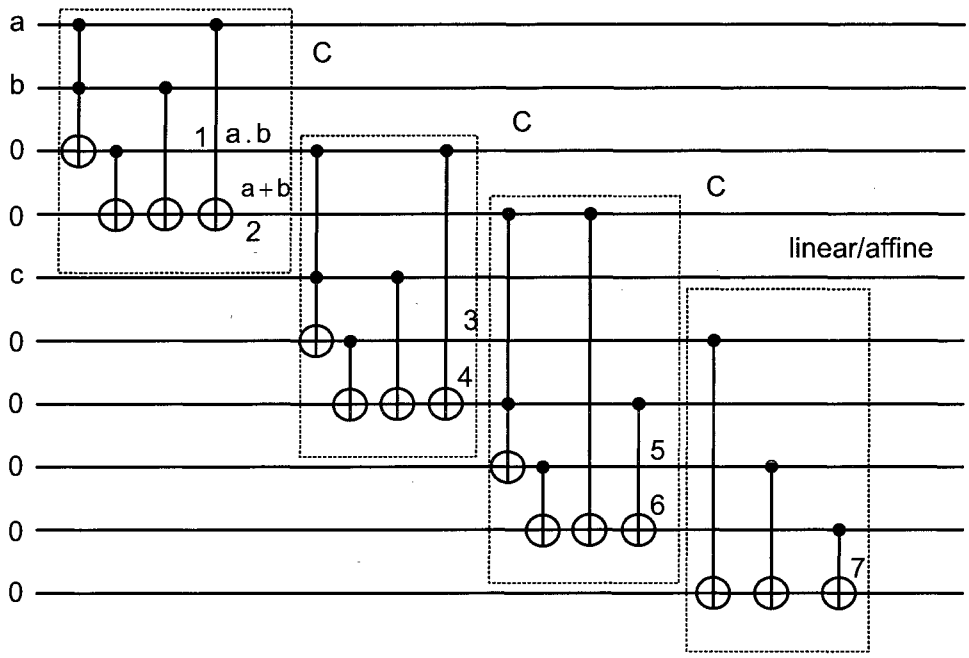


Figure 7.11.3.2: Standard quantum array (with dimension of time from left to right) for part of the reversible Net Figure 7.11.3.1.The blocks C are shown, the mirror circuit with C^{-1} are not shown.

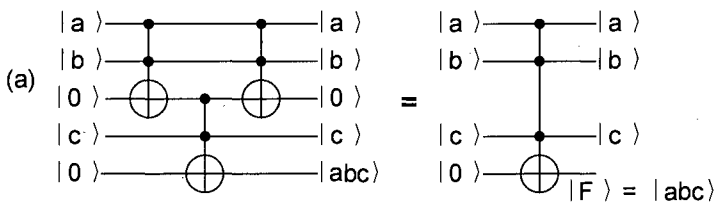
Each block C is the (non-reversible) min/max block where local output min = a • b and local output max = a + b. The reversible realization of the structure from Figure 7.11.3.1 is presented in Figure 7.11.3.2 (only half of the circuit, the mirrors are not drawn).

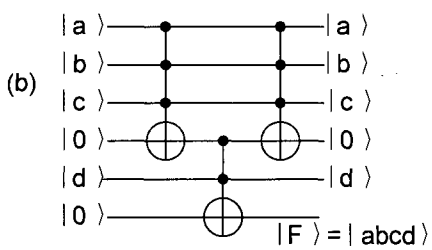
The mirror circuits in Figure 7.11.3.2. were not shown for simplification but they restore all primary inputs and constants in ancilla bits. The circuit in block "linear" in Figure 7.11.3.1 is in general an arbitrary multi-input and multi-output affine circuit which we will call "affine composition".

7.11.4. Realization of single minterm functions for functions of many variables.

In this section we will present realization of single minterm functions for functions of many variables. This topic is important as in each odd function a single one-minterm function has to be still realized in each of our synthesis methods.

Figure 7.11.4.1a presents the pattern of realizing all minterms (NPN class of $|abc\rangle$) of three variables. As we see two ancilla bits are necessary. Two methods of realizing NPN ($|abcd\rangle$) are shown in Figure 7.11.4.1b and Figure 7.11.4.1c.





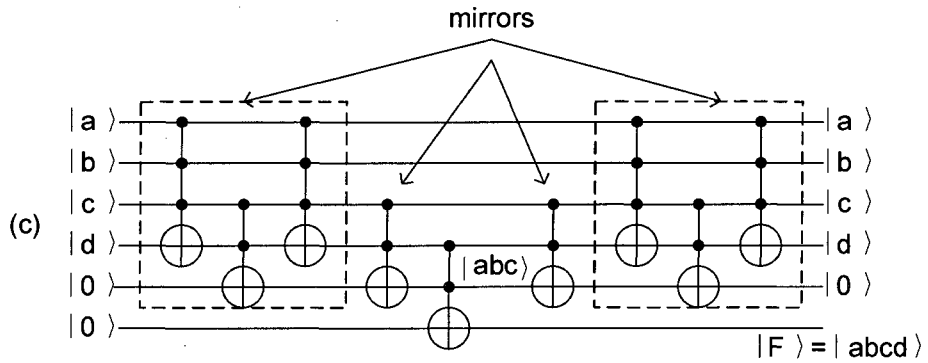
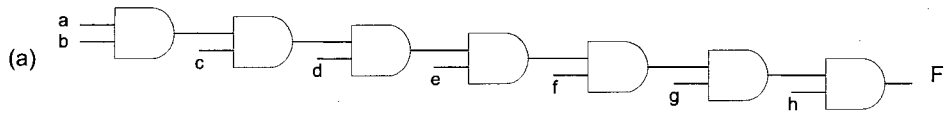
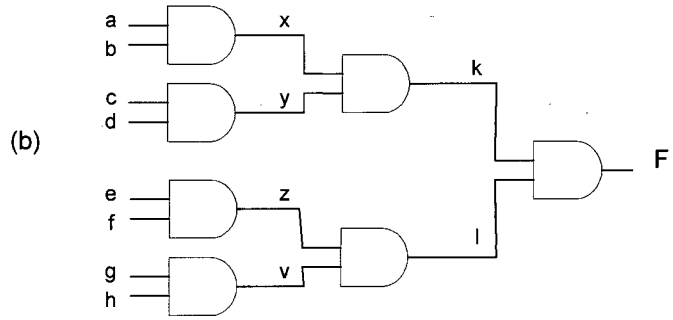


Figure 7.11.4.1: Recursive realization of big Toffoli gates. (a) Realization of F = abc using two ancilla bits and 3×3 Toffoli gates (next macro-generated to 2×2 quantum primitives), (b) realization of F = abcd using two ancilla bits, two 4×4 Toffoli is macro-generated as in Figure 7.11.4.1a, (c) another way to realize F = abcd using Toffoli gates. Although more expensive it may have some advantages as bits a b c d are neighbors.





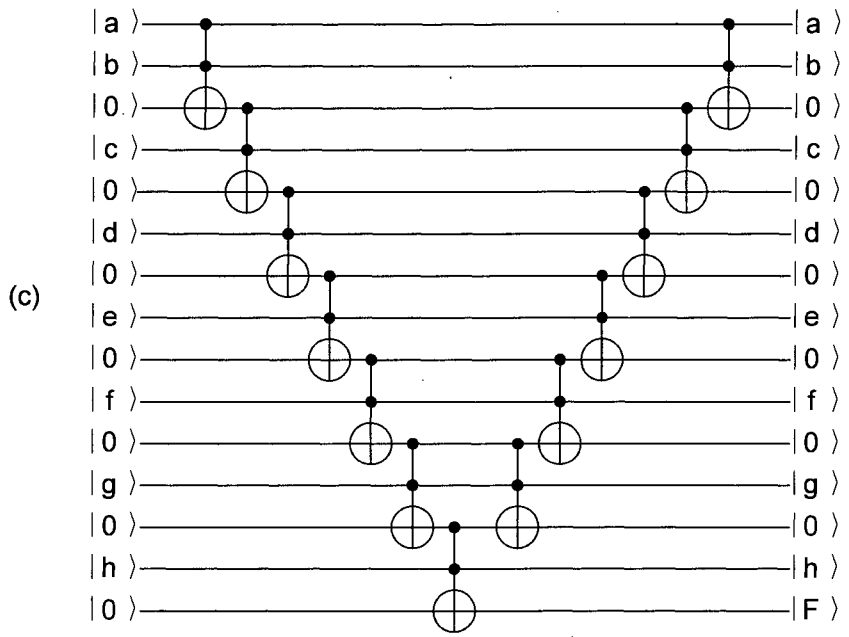


Figure 7.11.4.2: (a) Classical one-dimensional circuit for AND of many inputs, (b) classical tree circuit for AND of many inputs, (c) quantum circuit corresponding to circuit from Figure 7.11.4.2a has 13 3×3 Toffoli gates and 7 ancilla qubits.

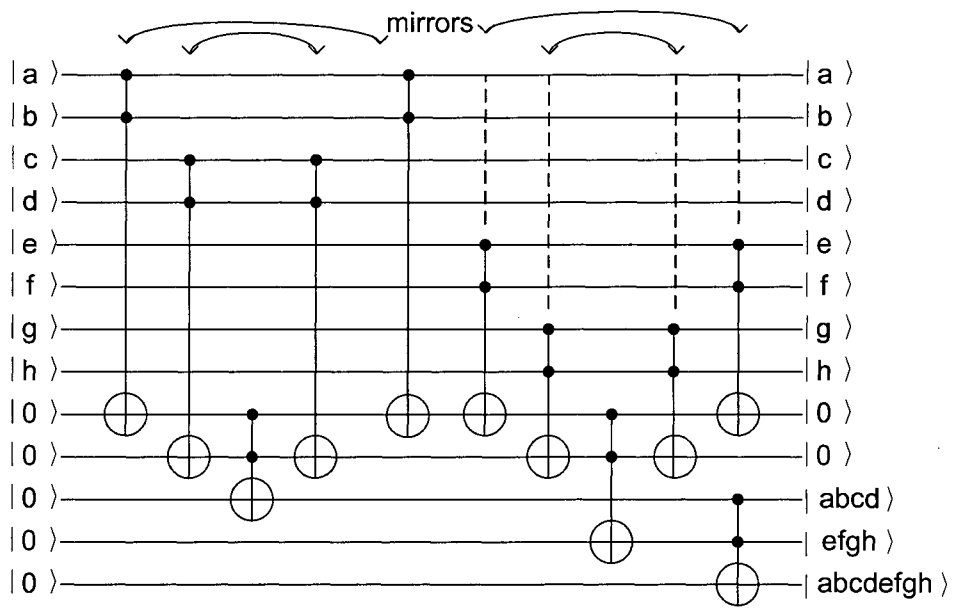


Figure 7.11.4.3: Reversible Folded variant of the circuit from Figure 7.11.4.2b. In general, the garbages $|abcd\rangle$ and $|efgh\rangle$ need mirrors to restore their qubits to constant $|0\rangle$.

The circuit from Figure 7.11.4.3 has 10 3×3 Toffoli gates and 5 ancilla bits but it has 2 garbages, $|abcd\rangle$ and $|efgh\rangle$. To remove these garbages the mirror of this circuit is necessary leading thus to the total of 20 3×3 Toffoli gates. Comparing to the circuit from Figure 7.11.4.2c we gain 2 ancilla bits at the cost of 20-13 = 7 3×3 Toffoli gates. What is better depends on the technology and on the fact if intermediate functions are reused in this or another output.

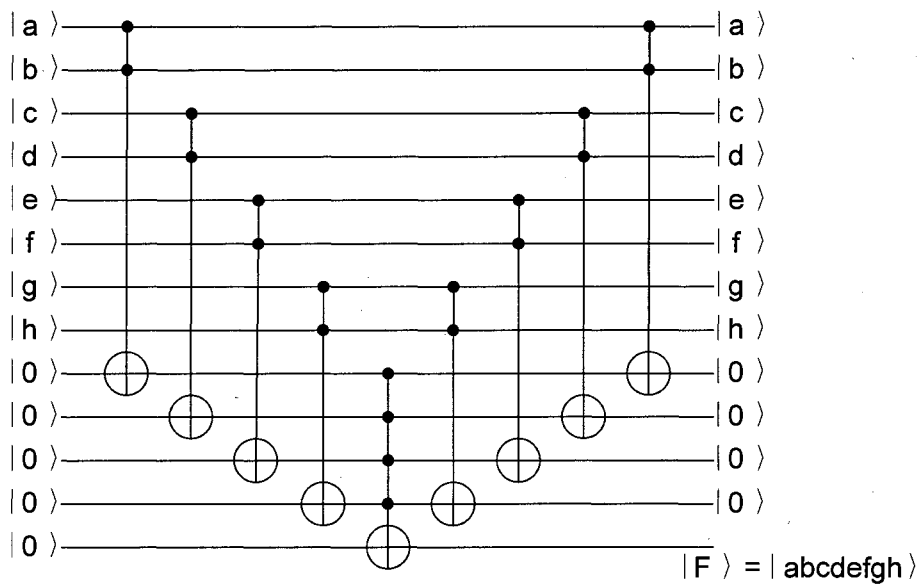


Figure 7.11.4.4: The quantum circuit for $|F\rangle = |abcdefgh\rangle$ with 5 ancilla bits, 8 3×3 Toffoli gates and one 5×5 Toffoli gate. No mirror circuit is needed in this variant.

7.11.5. Minterm Pair Functions.

Minterm pair functions are important starting points to affine complex gates and are a worst case solution to even functions of any number of variables. It is easy to prove

that every minterm pair in a function of 4 variables is realized in an affine Toffoli gate cheaper than realizing its both minterms separately (Figure 7.11.5.1).

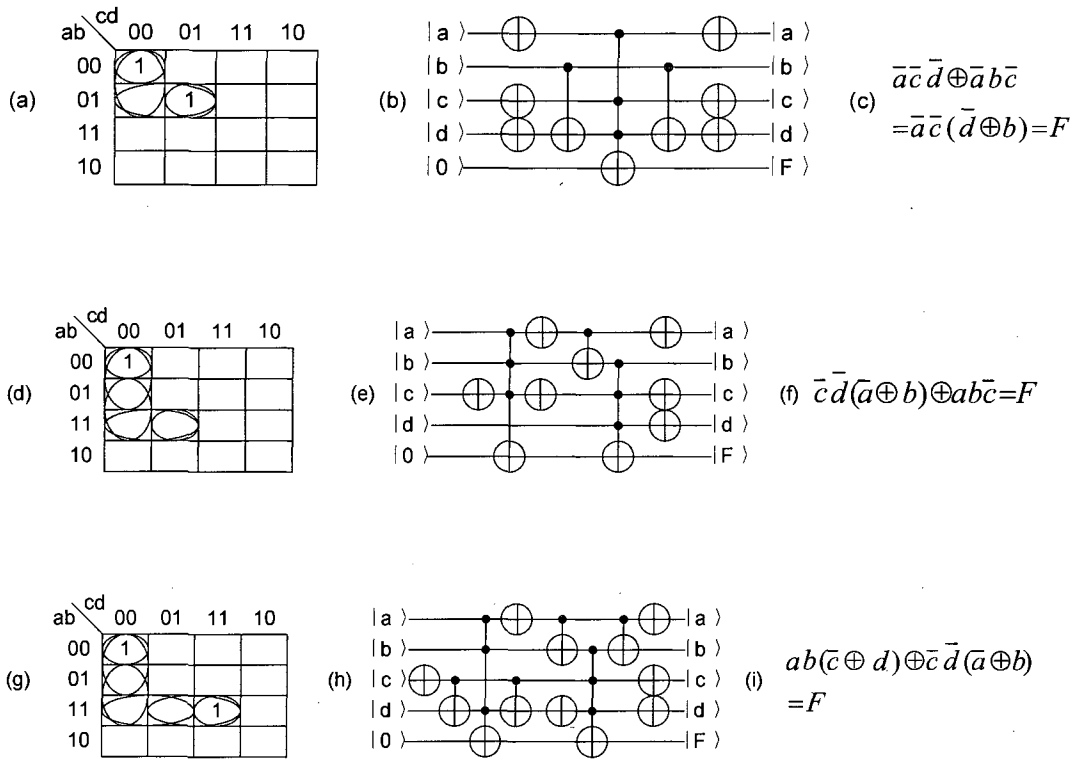


Figure 7.11.5.1: Chains in functions of 4 variables realized with affine Toffoli gates, (a) KMap for Hamming Distance = 2, (b) the circuit for HD = 2, (d) the KMap with HD = 3, (e) the circuit with HD = 3, (f) the formula with HD = 3, (g) the circuit with HD = 4, (i) the formula with HD = 4.

Next, because every function of 4 variables with 4 minterms can be built from pairs of minterms we prove that same for all functions with 4 minterms. Because of negation, the same result is true for function with 2^4- 4 = 12 minterms. Similarly it can be proved for all functions with 6 and 2^4- 6 = 10 minterms, see Figure 7.11.5.2.

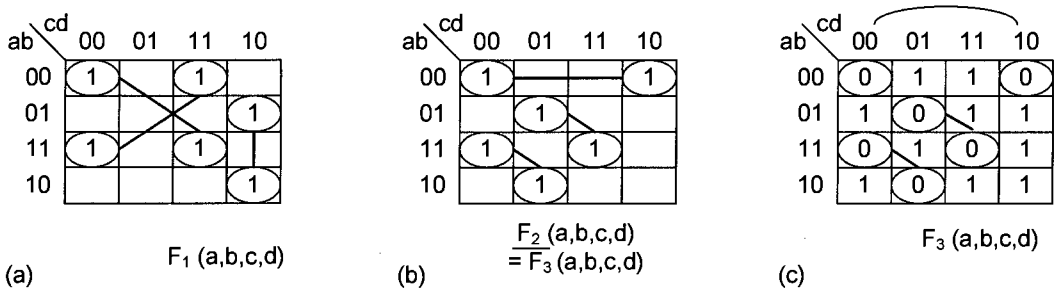
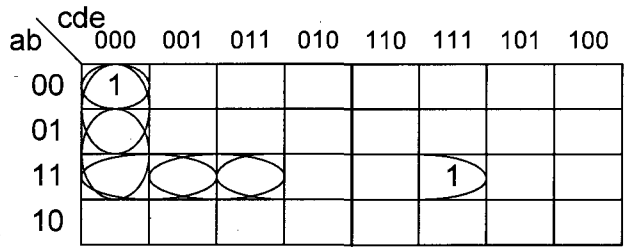


Figure 7.11.5.2: Functions with 6 and 10 minterms. Different decompositions of sets of minterms to minterm pairs. (a) decomposition to Affine Toffoli gate of 4 minterms and a minterm pair, (b) decomposition to three pairs of true minterms, (c) decomposition to three pairs of false minterms.

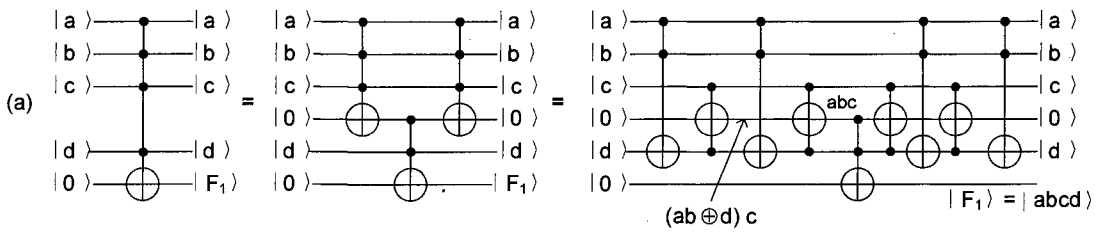
Finally the same is proven for functions (balanced) with 8 minterms.

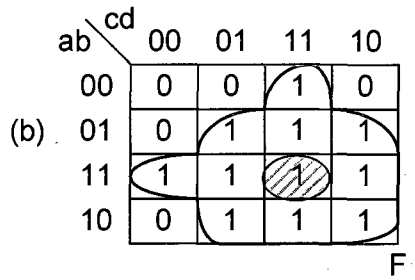


$$f = \bar{c}\,\bar{d}\,\bar{e}(\bar{a} \oplus b) \oplus ab(\bar{c}\,\bar{d} \oplus \bar{c}\,e \oplus de)$$

Figure 7.11.5.3: HD5 minterm pair function of 5 variables realized with 4 Toffoli gates and 2 Feynman gates.

Realization of NPN class representative function f for minterm pair functions of 5 variables is shown in Figure 7.11.5.3.





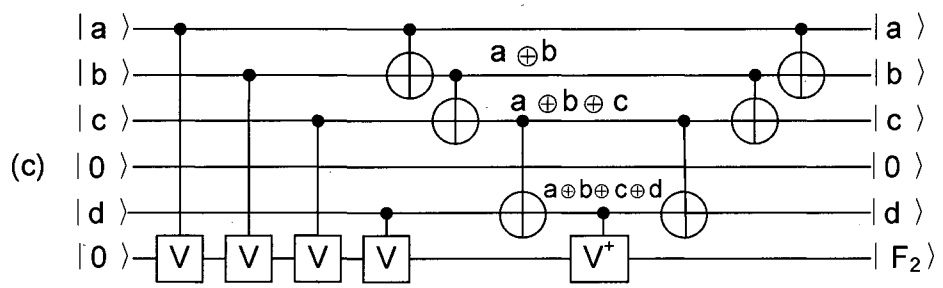
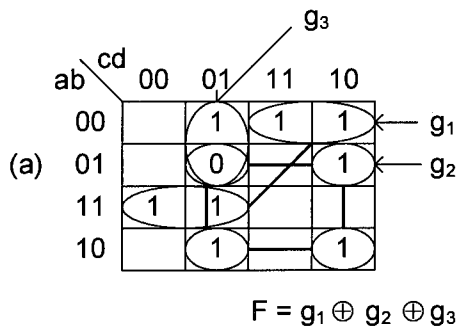


Figure 7.11.5.4: Explanation to composition (EXORing) of an irreversible function F to reversible functions F_1 and F_2, (a) realization of single-minterm" function $F_1(a,b,c,d)$ – its initial schematic and stages of circuit realization with smaller Toffoli gates. The gates at the right in Figure 7.11.5.4a should be further decomposed to 2×2 quantum primitives, (b) the original function F and its components F1(shaded) and F2 (in a loop), (c) realization of $F_2(a,b,c,d) = S^{2,3}(a,b,c,d)$ using only 2×2 quantum primitives.

Observe the same order of qubits in $|F_1\rangle$ and $|F_2\rangle$, a, b, c, 0, d, 0 that allows to ab at $|F_1\rangle$ and $|F_2\rangle$ without using SWAP gates. Example of using our decomposition method is shown in Figure 7.11.5.4. Function $|F\rangle$ was decomposed as $|F\rangle=|F_1\rangle\oplus|F_2\rangle$ and next

each of its component functions was realized and they were composed to one quantum array.



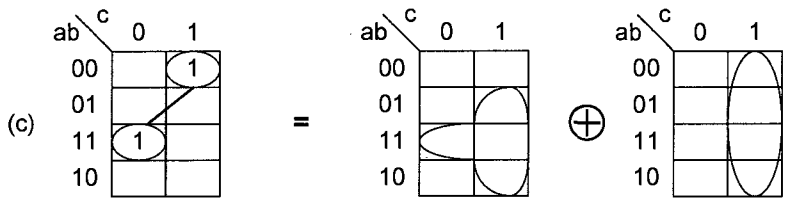
(b)

$$g_1 = \bar{a}\,\bar{b}\,c \oplus a\,b\,\bar{c} = maj(a,b,c) \oplus c$$

$$g_2 = \bar{a}\,b\,\bar{c}\,d \oplus \bar{a}\,b\,c\,\bar{d} \oplus a\,\bar{b}\,\bar{c}\,d \oplus a\,\bar{b}\,c\,\bar{d}$$

$$= \bar{a}\,b(c \oplus d) \oplus a\,\bar{b}\,(c \oplus d) = (a \oplus b)(c \oplus d)$$

$$g_3 = \bar{a}\,\bar{c}\,d$$



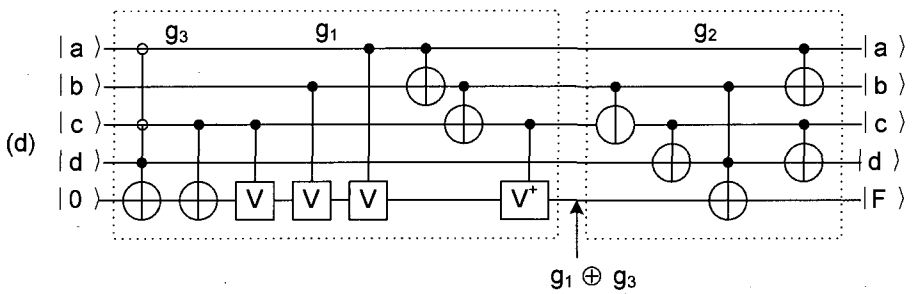


Figure 7.11.5.5:Example of decomposition to two ARNG functions and standard Toffoli gates, (a) the original function $F = g_1 \oplus g_2 \oplus g_3$ *where* g_1 *is an ARNG,* g_2 *is an Affine Toffoli gate and* g_3 *is standard Toffoli gate, (b) formulas for functions* g_1, g_2, g_3, *(c) realization of double-cube (minterm-pair) function* g_1, *(d) realization of* $|F\rangle$ *by composition of* g_1, g_2 *and* g_3.

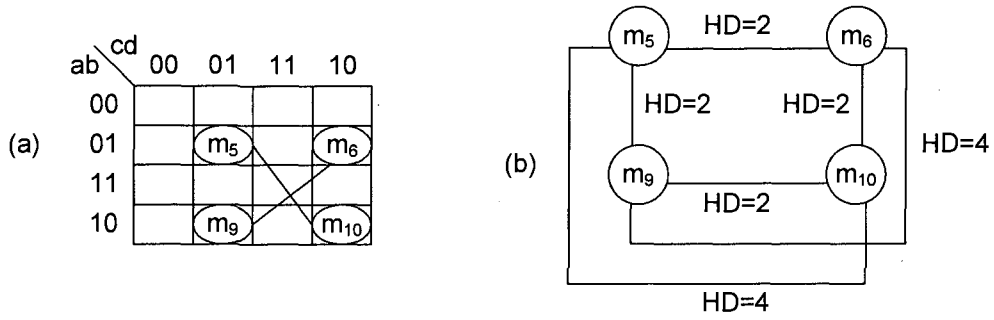


Figure 7.11.5.6: Visualization of affine patterns in KMaps of four variables, (a) 4 minterms of an affine group, (b) Hamming distances between all pairs of minterms from the pattern in Figure 7.11.5.6a.

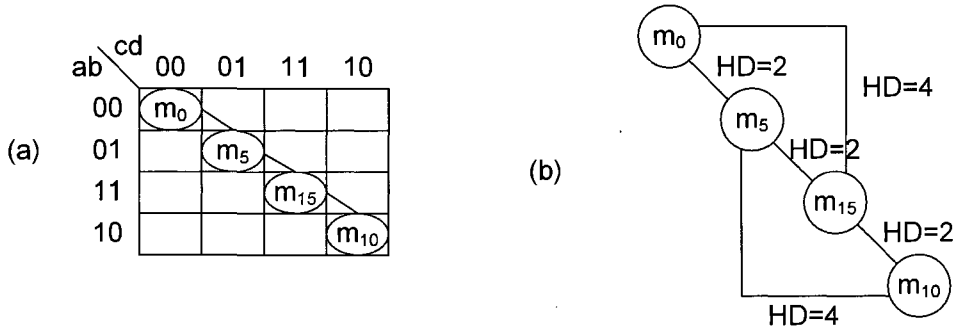


Figure 7.11.5.7: Visualization of affine patterns in KMaps of four variables, (a) four minterms from an affine group $\bar{a}\bar{b}\bar{c}\bar{d} \oplus \bar{a}b\bar{c}d \oplus abcd \oplus a\bar{b}cd = \bar{a}\bar{c}(\bar{b}\bar{d} \oplus bd) \oplus ac(bd \oplus \bar{b}\bar{d})$ $= (\bar{a}\bar{c} \oplus ac)(\bar{b}\bar{d} \oplus bd) = \overline{a \oplus c} \bullet \overline{b \oplus d}$ *, (b) Hamming distances between all pairs of minterms from the pattern in Figure 7.11.5.7a. Please note isomorphism of graphs in Figure 7.11.5.6b and Figure 7.11.5.7b.*

Another example of decomposition to ARNG and Toffoli is presented and explained in details in Figure 7.11.5.5. Visual patterns of Affine Toffoli gates are given and compared in Figures 7.11.5.6 and 7.11.5.7.

Concluding:

1. My method improves on $3/4^{th}$ of Toffoli-like functions of 2 variables (3 out of 4).
2. Balanced functions of 3 variables. There are the following function classes: Exors of 2 literals (6 functions), Exors of 3 literals (2 functions), majority (8 functions), Toffoli-like/Davio-like (24 functions), multiplexers (Shannon-like) – 24 functions. A total of 64 out of 70 balanced functions (there are 3*2 = 6 binary literal functions which are balanced but not interesting). My method improves majorities (8 functions), $3/4^{th}$ of Toffoli-like functions (18) and $3/4^{th}$ of Shannon like functions (18). Thus 44 out of 64 balanced functions are improved by having a smaller quantum cost.
3. There are 8!/(2! 6!) = 28 functions (non-balanced) functions of 3 variables with 2 minterms. They can be all realized by double-minterm gates thus there is an improvement on 28 functions with 2 minterms.
4. There are 8 non-balanced functions being single minterms. No improvement exists on these functions.
5. There are 28 functions of 3 variables with 6 minterms. They are negations of functions from point 3 above. There is improvement on all of them.
6. Similar results exist for functions of four and five variables but there is no space here to perform a complete analysis.

Concluding on this method, please note, that our method uses special functions and gives special advantages for special functions but in real-life problems such functions rather than random function, frequently occur. This fact was observed for the first time by the developers of the very successful SOP minimizer Expresso.

7.12. Conclusions on affine concepts and decompositions.

Because this chapter is the core of innovative ideas from this thesis, and its conclusions are important to the entire dissertation, we will write these conclusions in points.

1. Designing an optimized cascade of reversible ("quantum permutative") gates is one of the fundamental problems in quantum computing, because such cascades are used in logic blocks in oracles of Grover algorithm, in the arithmetic part of the Shor algorithm and in other quantum circuits and algorithms. Therefore designing a method to improve on cost and speed related factors of such cascades has a fundamental importance to quantum circuit design.

2. In this chapter we introduced the concept of affine gates, and we showed examples of applications of this concept to design binary quantum gates called

affine root-of-NOT gates and affine Toffoli gates. This way the realization costs of some permutative functions are dramatically reduced.

3. We introduced also the concepts of affine polarity and canonical affine forms that generalize the Fixed Polarity Reed-Muller forms. This way the concept of affine functions is extended to the minimization of arbitrary functions, (although not necessarily the optimal results are always obtained). This is a theoretical contribution with possible practical applications.

4. We showed also other ways of using our new concepts in Boolean logic synthesis. The examples were able to show always the reduced quantum costs.

5. Combinations of various binary permutative gates and synthesis methods can be used to create the new types of quantum cascades as introduced in this chapter. The choice of the gate types and their realization using quantum-realizable primitives are thus of basic importance to binary quantum logic synthesis algorithms.

6. We wrote the program generator CircuitSearch for exact minimization of affine-controlled V, $V^{\dagger}$ and NOT gates and circuits to be used in various types of reversible cascades. Because this program is based on exhaustive search, we tried several search strategies to make the program as efficient as possible. The

iterative deepening depth-first search method is more practical for these tasks than the biology-mimicking methods such as the genetic algorithm. The full potential of Iterative Deepening has been not yet fully recognized in quantum circuit's community. It can be combined with A* search algorithm by adding a heuristic evaluation function. There are further possibilities of improving CircuitSearch which were not yet investigated.

7. The combination of CircuitSearch and circuit decompositions presented here for many benchmark functions dramatically reduces quantum costs of reversible cascades. It is important to observe that in all classical technologies many-input AND gates can be built rather inexpensively. However in quantum technologies, because of the necessity to build from only 2*2 primitives, the multi-input AND gates belong to the most expensive functions to realize. Therefore the synthesis methods should not be based on exoring ANDs, but on exoring some other basic functions, that are inexpensively realizable in quantum technology. Some of such functions were introduced in this chapter.

8. The methods presented in this chapter allow to investigate trade-offs between the number of gates and the number of the ancilla bits. For instance, a circuit without ancilla bits may be theoretically realizable but would likely be much longer than a circuit with one ancilla bit. The related question of synthesis is a difficult one and open to future research.

9. An interesting open problem is to extend these ideas and search methods to arbitrary radix logic, for instance ternary. The first attempt to do this can be found in Chapter 10.

10. When we get experience with CircuitSearch we found the properties of functions that can be minimized by this approach efficiently. This experience suggested us to design the library of inexpensive gates and to create the new methodology of decomposing large functions to small functions that use affine methods (in one or another way).

Although the synthesis of multi-output quantum arrays is beyond the scope of this thesis, we found that the methods presented in this chapter are useful in several practical multi-output function minimization problems. Thus we used these methods for every output separately and we tried to reuse some subfunctions such as affine Toffoli or affine other types of gates in the synthesis processes of multi-output functions. There will be more designs of this type in Chapter 11 but now let us realize, as a multi-output function, the comparator (A=B), (A>B), (A<B) of two two-bit numbers. The same method as explained below can be used to any size of this type of a comparator.

Example 7.12.1:

We will be designing a comparator realizing together three predicates (A = B), (A > B), (A < B). Such comparators have many applications (Chapters 11, 13 and 14). We assume that both signals A and B have two bits each. The KMap of functions (A = B), (A > B), (A < B) are given in Figure 7.12.1 a, b and c, respectively. Observe (as shown in Figure 7.12.2) that these signals are disjoint and complete, which fact suggests the synthesis method to be selected. We assume thus to realize two of the three functions and create the third one by subtracting their sum from the logic unity of four variables a, b, c and d. Because of symmetry of functions (A > B) and (A < B) any of them can be selected for realization; we select (A > B). It is found that function (A = B) is the Affine Toffoli gate, realized as in Figure 7.12.3. Any subfunction of this function can be now reused to synthesize other functions. The function (A > B) can be decomposed to an EXOR of a product " a c' " and a double-minterm function 0100 ⊕ 1110 = a' b c' d' ⊕ a b c d' . The double minterm is realized as an Affine Toffoli gate and the term ac' is realized as a classical Toffoli gate. However, in the next stages the Toffoli gates and the Affine Toffoli gates are replaced with their realizations based on synthesis of big gates and on CV/CV† based synthesis methods. This allows to decrease the quantum costs. Synthesis of the predicate function (A>B) is using Toffoli and Affine Toffoli gates is shown in Figure 7.12.4.

The final circuit, that reuses block (a ⊕ c)' and uses mirror gates to restore initial states of input variables, is shown in Figure 7.12.5. Finally, by replacing all 3*3 and

4*4 Toffoli gates with 2*2 quantum primitives we obtain the circuit from Figure 7.12.6. This example showed the essence of methods from this chapter and how they can be combined in multi-output quantum circuit synthesis.

Although the final circuit with 2*2 quantum primitives may look expensive, its quantum cost in terms of 2*2 quantum primitives is dramatically smaller than that of a circuit synthesized using traditional methods of quantum synthesis such as MMD, Agrawal/Jha or Mishchenko/Perkowski approaches. In these approaches every output function would be realized separately as an ESOP. This would require four 5*5 Toffoli gates for (A=B), one 3*3 Toffoli gate and two 4 *4 Toffoli gates for (A>B), and one 3*3 Toffoli gate and two 4 *4 Toffoli gates for (A<B) (even if the best EXOR cover solutions were found for the (A<B) and (A>B) predicates. Observe that the circuit from Figure 7.12.5 has only one 4*4 Toffoli gate and two 3*3 Toffoli gates. These gates in any case constitute much higher part of the total quantum cost of this circuit than the NOT and CNOT gates being the affine components.

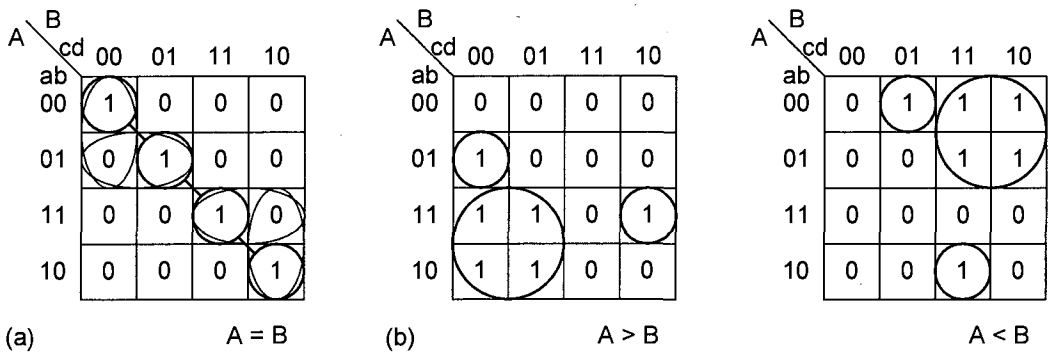


Figure 7.12.1: Specification of the problem of designing a comparator with three predicates. (a) Kmap for two-bit arguments for function (A=B), (b) Kmap for two-bit arguments for function (A>B), Kmap for two-bit arguments for function (A<B).

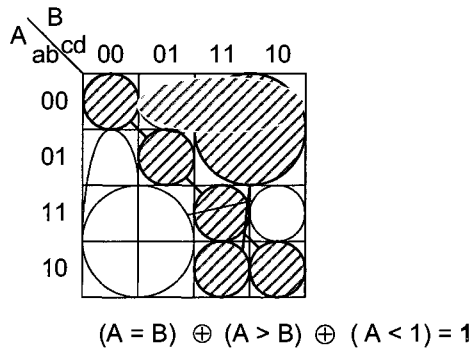


Figure 7.12.2: Specification of the problem of designing a comparator with three predicates. Kmap illustrates the disjointness and completeness of predicate functions (A=B), (A>B), and (A<B).

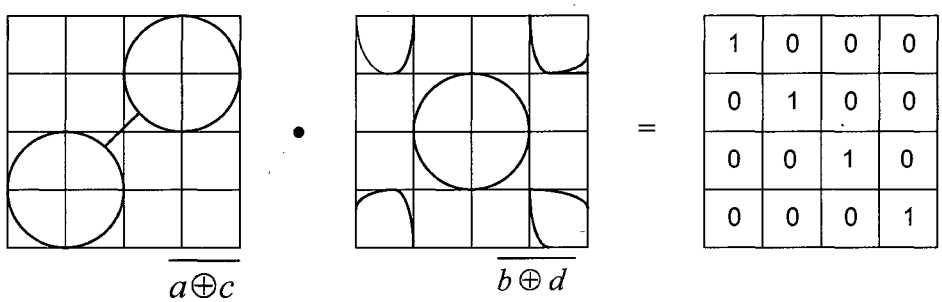
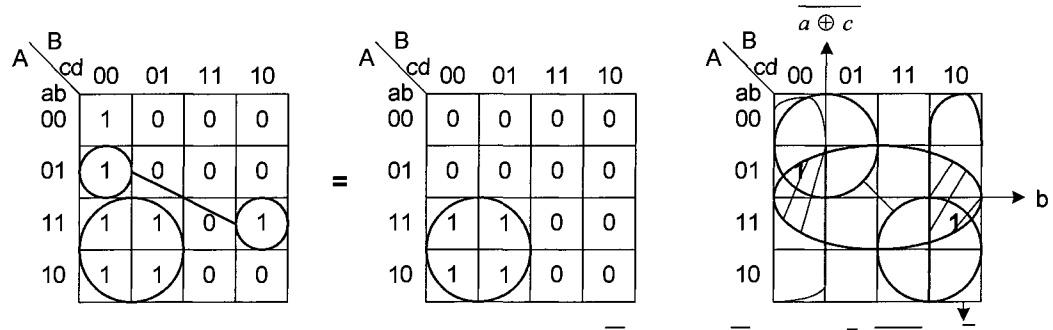


*Figure 7.12.3: Graphical illustration for the realization of Affine Toffoli gate (a ⊕ c)' * (b ⊕ d)' for predicate function (A=B).*



$$(A > B) = a\,\bar{c} \oplus (\bar{a}\,b\,\bar{c}\,\bar{d} \oplus a\,b\,c\,\bar{d})$$
$$= a\,\bar{c} \oplus b\,\bar{d}\,(\bar{a}\,\bar{c} \oplus a\,c)$$
$$= a\,\bar{c} \oplus b\,\bar{d}\,\overline{a \oplus c}$$

Figure 7.12.4: Graphical illustration for the realization of composition of Toffoli and Affine Toffoli gates $a\bar{c} \oplus b\bar{d}$ *(a ⊕ c)' for predicate function (A > B).*

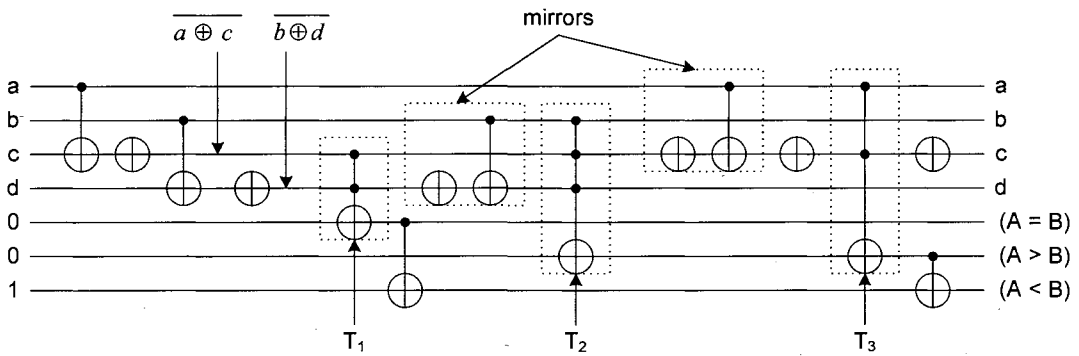


Figure 7.12.5: The quantum array for the complete three-output comparator circuit realized in Example 7.12.1. Please note the role of mirror gates used to restore original input values to be reused in other gates and the order of gate realizations to allow multi-output design with the reduced number of gates and ancilla bits. The lowest ancilla qubit is initialized to value 1 to make use of disjointness and completeness of functions (A=B), (A>B), and (A<B).

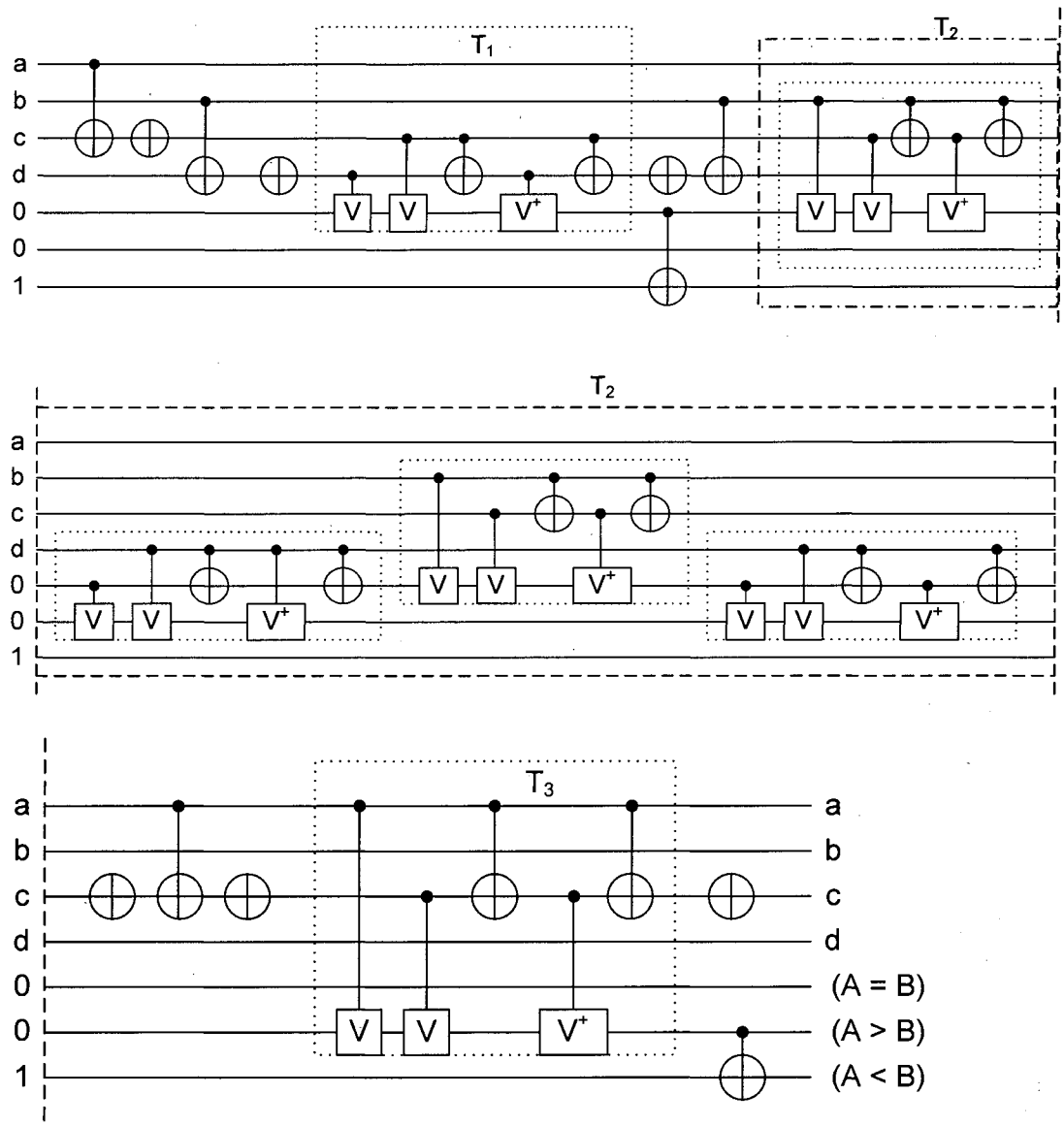


*Figure 7.12.6: The quantum array for the complete three-output comparator circuit realized in Example 7.12.1 and in Figure 7.12.5. Please note the use of only 2*2 quantum primitives and inverters. Observe how expensive is the quantum realization of the 4×4 Toffoli gate T_2.*

The example above assumed several tricks used together to minimize the function. This leads to dramatic improvements. However we found that many arithmetic and

other functions from real oracles have the desired properties that allow to synthesize them with small quantum costs even without executing a lot of search.

Concluding on affine functions and gates introduced in this chapter. Although the new methods presented in this chapter cannot improve the synthesis of every function, they improve the designs of many functions, including the practically important functions used in arithmetic, logic, predicate, comparison, spectral and other blocks used as parts of oracles in Grover Algorithm and in Shor Algorithm (see Chapter 11). For instance, the methods do not improve realization of a single product of many variables, but in such cases the methods, as shown in this chapter, can be used together with well-known methods to decompose big gates to 2*2 quantum primitive gates. Thus, combining the new methods with the synthesis methods developed previously but not used in automatic synthesis algorithms so far, the combined methods proposed in this chapter improve on the realization of every single-output or multi-output Boolean function evaluated using quantum costs that were introduced in chapter 2.

CHAPTER 8

Minimization of Incompletely Specified Boolean Functions for Generalized Reed-Muller Forms realized in Quantum Arrays

8.1. Introduction

Past experience has shown the GA applied to logic minimization had limitations of size, computation time, and solution optimality [Dill97, Dill97a]. In comparison, several decades of research have contributed to the current human understanding and efficient implementation of systems for logic design and minimization. As presented in literature the AND-EXOR circuits have been shown economical and easily testable [Biamonte05, Pradhan87]. They have the nice property of the *"more ones than zeros"* group selection heuristic that can improve sequential choices of terms in greedy and search algorithms. AND-EXOR circuits have another nice property of extracting linear variables, finding linear pre- and post-processors, and polarity selection for canonical expansion forms. As discussed in Chapters 3, 4 and 7 the AND-EXOR circuits are also a natural match to quantum arrays and require much smaller number of ancilla qubits than the AND-OR logic. Hence in this manner, the minimization techniques presented in Chapters 3, 4 and 7 may be expanded, in the future, for multi-valued logic hardware or data mining applications (some preliminary ideas can be found in Chapter 10).

As we remember, practically all problems of our interest can be solved by search, and thus can be solved either by a special search algorithms like in Chapter 7 or by our general search mechanism from Chapters 5 and 6. Thus the extended cybernetic search approach from the Chapter 6 can be applied to several applications in quantum circuit design. It is also the base of quantum search algorithms from chapter 6 illustrated practically in chapters 12 – 15. This method is applied in Chapter 8 to the minimization of incompletely specified functions in the quantum array that realizes the Generalized Reed-Muller Forms (single and multi-output). In contrast to most methods from the literature (except for Bruce Yen [Yen05]) this algorithm not only minimizes the reversible circuit but also performs the conversion from a non-reversible to a reversible circuit. The presented work improves on several previously published papers in the area, especially on the heuristic search-based work of Sasao and Debnath [Sasao94] and the GA-based work of Dill and Perkowski [Dill97, Dill97a]. The developed in Chapter 8 original automated technique for logic minimization of incompletely specified data Generalized Reed-Muller Forms is based on generalized search processes presented earlier and a multi-strategic approach is taken. Human expertise is combined with the extended cybernetic search mechanism, for the development of an efficient problem-solving expert system. This method formalizes the *"hand and eye"* minimization methods outlined in Chapters 2, 3, 4, 6 and 7.

8.2. Generating systematically all product terms for all GRMs of all polarities and related problems.

There are 3^n products of literals for a function of n variables. All these products can be visualized by ternary hypercube, using Ternary Gray Code. This is a new representation idea that has been not investigated so far. The space of all literal products for functions of 2 variables is presented in Figure 8.2.1. The Hamming Distance One (HD1) path through this space which is shown in Figure 8.2.2. This path is not closed.

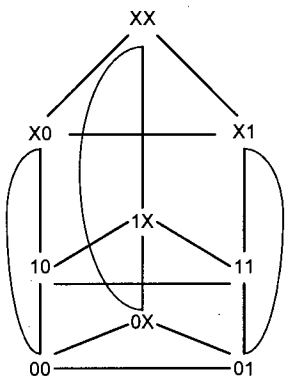


Figure 8.2.1: Space of generalized polarities for 2 variables using Ternary Gray Code. Every edge is for Hamming Distance 1 nodes.

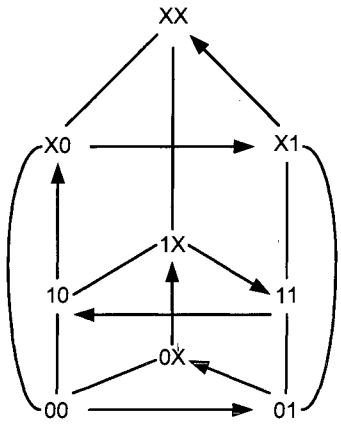


Figure 8.2.2: A Hamming-Distance-1 path in the generalized polarities space corresponds to ternary Gray code counting (this is an open path).

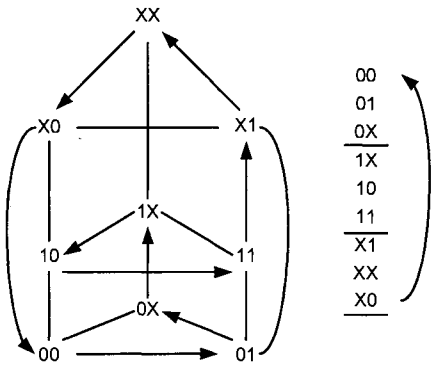


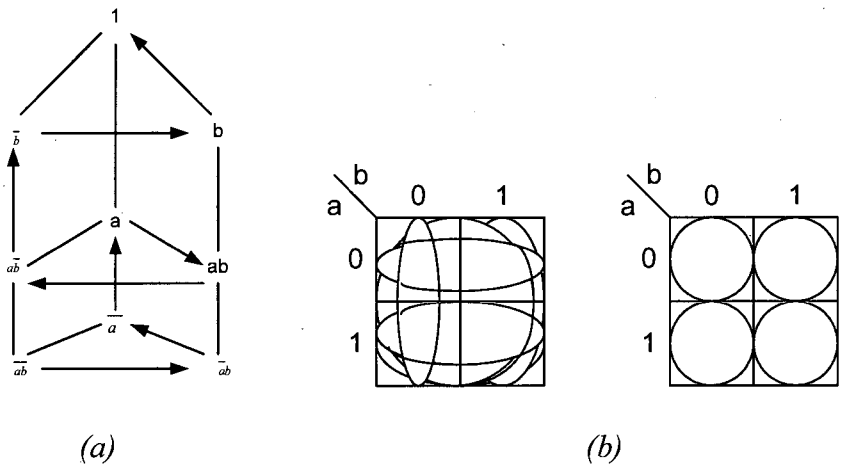
Figure 8.2.3: A Hamming-Distance-1 path in the generalized polarities space corresponds to ternary Gray code counting (Closed Ternary path). The path is generated as the ternary Gray counting sequence shown at the right of the Figure.

Figure 8.2.1 presents the space that represents the Generalized polarity. The path of through this space in which all subsequent nodes have Hamming Distance of 1 is shown in Figure 8.2.2. In the graph from Figure 8.2.2 we can start from a node with

polarity 00, then go to node with polarity 01 and then to node 0X. Here X polarity means non existence polarity or no polarity (as a variable not taken into account to this generalized polarity). Our algorithm works as an incremental counter as shown at the right of Figure 8.2.3. This variant shows another path that is a loop. A loop is a path that is closed, it is presented at the left of Figure 8.2.3.

Example 8.2.1:

All literal product groups of all GRM forms generated for two variables are the following: 1 , $\overline{b}$, b , a , $\overline{a}$, $a\,\overline{b}$, ab , $\overline{a}\,\overline{b}$, $\overline{a}\,b$. Of course, one can calculate that there are $3^n = 3^2 = 9$ such products.



(a) *(b)*

Figure 8.2.4: (a) Hamming-Distance-1 path of all groups generated for all GRM polarities for two variables. (b) All groups generated for GRM polarities using a KMap.

Figure 8.2.4(a) shows all groups generated systematically as the path follows $\bar{a}\bar{b}$, $\bar{a}b$, $\bar{a}$, a, ab, $a\bar{b}$, $\bar{b}$, b, 1. Figure 8.2.4(b) presents two 2-variable KMaps of all possible groups; that means we are creating all possible products of literals. For GRM, we have 3^n of all possible groups. Now, if we have GRM for 2 variables, we have 1, a, b and ab. We can negate or NOT and it can be omitted in the group. So, it is either omitted, which is X or a' for 0 and 1 for a, for this one. For instance, if we go from $\bar{a}\bar{b}$ to $a\bar{b}$, we are changing the polarity of one variable. Every variable, we can negate or NOT. This is the way in which we systematically generate all possible product term groups which exist in all possible GRMs.

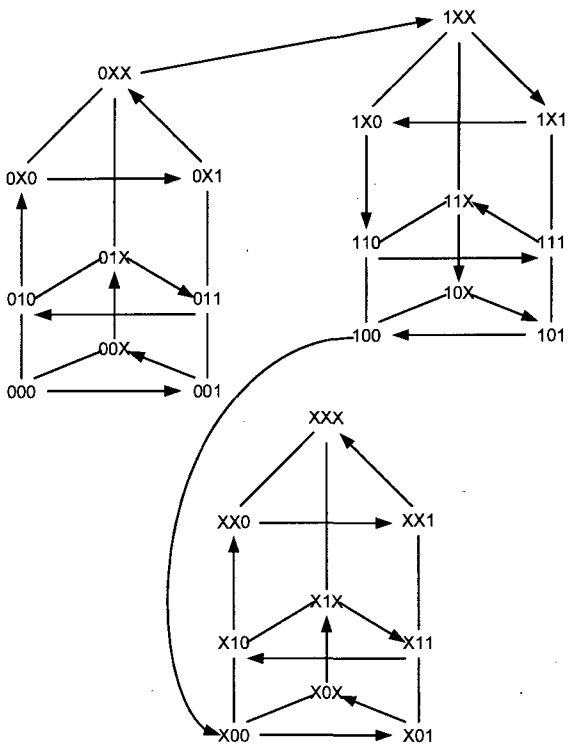


Figure 8.2.5: Three Dimensional Space of generalized polarities for functions of 3 variables using Ternary Gray code.

	110
000	111
001	11X
00X	10X
01X	101
011	100
010	X00
0X0	X01
0X1	X0X
0XX	X1X
1XX	X11
1X1	X10
1X0	XX0
110	XX1
	XXX

Figure 8.2.6: Ternary Gray Code counting for generalized polarities. This way of counting corresponds to Figure 8.2.5.

Figure 8.2.6 illustrates the way of counting, 000, 001, 00X, etc. The counting is done by increasing by one a number contained in a Ternary Counter. The order of enumeration in every bit of the counter is here 0, 1, X. The counting sequence assumes that X is the highest value. Thus reaching X we have to increase in the next bit, hence 00X will increase to the polarity of 01X. The counting should be from node to node in the whole space, all counting should be in Gray code, in HD distance one and has edge. This is called the Ternary Gray code counting for generalized polarities. This is just one way of systematic generation of all product groups. In Figure 8.2.5 given is the three dimensional visualization for the algorithm that will create systematically all possible generalized polarities in certain (Gray code) order.

Observe that the literal product groups generated as explained in this section can be used not only in GRM but also in ESOP and other circuit types.

8.3. The Extended Cybernetic Search used to solve the GRM minimization problem

In this research, the ECPS system from chapter 6 was employed. The general structure of search is shown in Figure 8.3.1. Next sections of this chapter will present some details and variants of implementation of this general idea.

In known algorithms, polarity strings are generated for GRM forms. Following the generation of a population of polarity strings, several iterations using the heuristic logic minimization method construct some of the possible GRM expressions (which have the polarity described by their associated polarity string) that represent the incompletely specified data set.

The polarity is the binary string representing the genotype in an evolutionary algorithm. The GRM form is the expression representing the phenotype. Note that for a completely specified function there is a one-to-one mapping from the polarity to the GRM form. In contrast, for the incompletely specified function, there are many GRM expressions corresponding to any given polarity. Thus, our "two layer search" ECPS-like algorithm heuristically selects one of many phenotypes corresponding to the given genotype. The best results (i.e. the GRM equations with the fewest terms) from several iterations of the heuristic method are then selected.

What does it mean "the best results"? The best results are those that minimize the value of the cost function.

The cost function for our algorithm is one of the following:

1. The total number of terms, in each of the best GRMs, for each of the multiple outputs of the function. Thus each output cost is calculated separately.
2. Any of the quantum costs introduced in Chapter 2.

The summation of the cost functions from each of the output functions (counting the duplicate terms only once) is then associated with the polarity as the fitness value for this polarity in the GA. The second cost function is NMR-technology-related. It can be a number of 2×2 gates or a number of NMR pulses.

Following the assignment of fitness values to choices based on the logic minimization heuristics, the GA proceeds with the standard search process (see Figure 8.3.1).

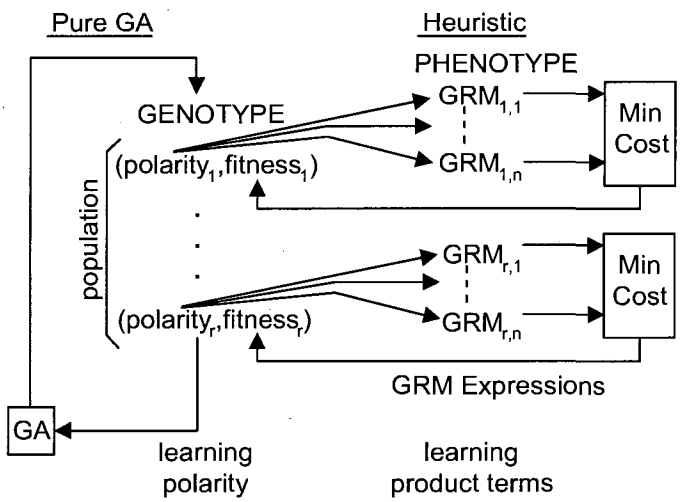


Figure 8.3.1: The general idea of hierarchical search applied to GRM forms for incompletely specified functions. The upper level – a GA selects the polarity and the lower level – the heuristic search selects the best circuit for the given polarity.

The polarity strings genotypes are similar to chromosomes in a standard GA. These strings are essentially the constraints for the selection of the explicit GRM solutions

(phenotypes). The GRM phenotypes, which describe the complete problem solution, are learned in the environment of GRM expressions. In this model, the learned behavior is simulated by the application-specific heuristic. There are several heuristics that we tested, but they are all based on the well-known "more ones than zeros" group selection principle. Herein, the ECPS minimization algorithm acts as a local heuristic search mechanism, deriving optimum GRM circuits given their polarity vectors. Because the fitness function is related to the total final cost of the multi-output circuit, the parameters of the polarity vector chromosome (genotype) and the fitness are indirectly linked to each of the GRM forms (the phenotypes). In the ECSPS search environment in the algorithm that creates new minimal GRMs, the polarity vector *"chromosomes"* remain unaltered during the local cost-minimizing search. (This remark about ECPS relates particularly to the ECPS-GRM variant presented here. ECPS can be also used in a different way). Then through the upper layer search process, new polarity vectors are created. This hierarchical and heuristic search process is illustrated in Figure 8.3.1. The reader should keep in mind that this is only a general scheme, out of which many detailed variants of search in ECPS can be created.

The systematic way to create all generalized polarities for a GRM of 3 variables a, b, c is given in Figure 8.3.2 (few polarities only shown).

Variants that can be programmed.

Concluding, based on the above ideas, the following different new approaches are possible to minimize incompletely specified functions using canonical forms such as FPRMs, GRMs and other canonical AND/EXOR forms.

- Method 1. Go through all polarities, use search for the best subset of product terms in each polarity.
- Method 2. Use GA to find the polarities and next use the greedy probabilistic search for each polarity (this is implemented in our approach from section 8.4).
- Method 3. Generate all GRM product terms as in section 8.2 in the order from the least expensive to the most expensive terms. For each group calculate the ratio of true minterms to false minterms (this ration is just a one particular variant of the "more ones than zeros" heuristic). Select the best groups, one from each polarity, and iterate with other choices to improve the result.
- Method 4. To find the best GRM, first find the best FPRM and next generate GRMs from it (see Figure 8.3.3 for the explanation). The search is not exhaustive.

All these methods can be programmed within the ECPS framework and compared. Only a partial comparison was done in this thesis.

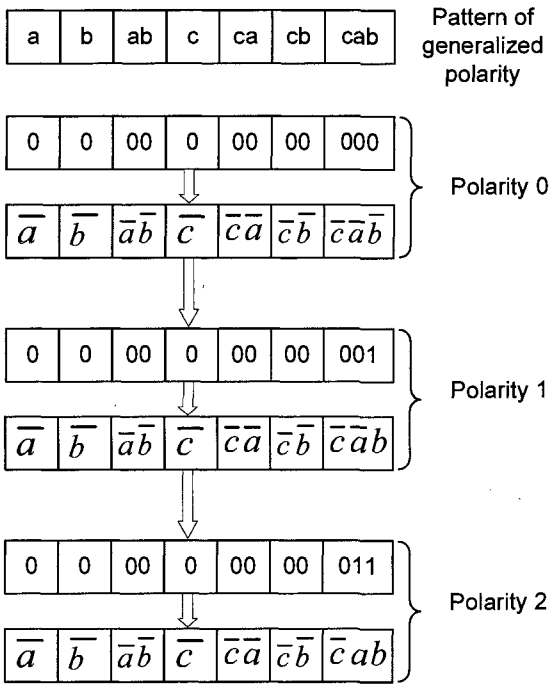


Figure 8.3.2: A systematic way to create all polarities for a function of three variables: a, b, c. Each generalized (GRM) polarity is represented by a binary string and the Gray code (HD1) enumeration is used. Generation of only 3 polarities is given here. This method can be used to generate all polarities or any subset of them to be used in an algorithm from algorithms in "Variants that can be programmed" above.

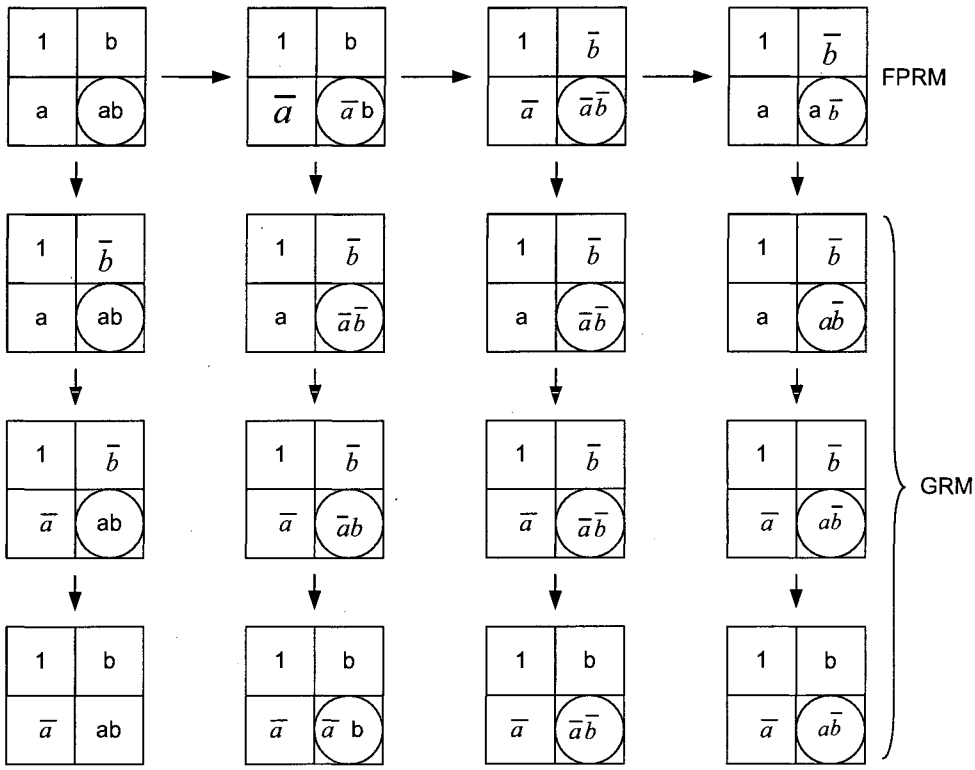


Figure 8.3.3: Maps for another approach (Method 4) for systematic creation of all GRMs for functions of two variables. This graph realizes the entire space of all GRMs. It can be also searched in a greedy way by first going horizontally through all FPRMs to find the best one and next start from it and go vertically down.

8.4 Illustrative Example of Minimization for Incompletely Specified Fuction Specification with GRM Forms

8.4.1. Introductory Examples

An example of the minimization heuristic for incompletely specified data with the GRM form of selected polarity is first given, followed by the complete algorithm description in Section 8.5. The principle of this algorithm is to consecutively select

product groups, denoted as g_i, and exor them from the function. Thus, recursively using the logic principle, $f = g_i \oplus f_{tail} \Rightarrow f_{tail} = g_i \oplus f$, the realization of the function f is obtained as an EXOR of all selected g_i product groups. This algorithm belongs to the "subset selection" family of algorithms with the "more ones than zeros" heuristics. This algorithm uses the well-known from Chapters 2, 3, and 6 greedy method of solving the even-odd covering problem that works well when a good heuristic for selecting consecutive product groups (g_i) is provided. Herein, the heuristic selection of cubes is based on minimizing the cost function using the variant of the *"more ones than zeros"* heuristics. In this particular approach the additional constraint also exists that all product groups g_i are consistent with the polarity vector (i.e. with the genotype). It means the group of a polarity different that the current polarity vector cannot be selected.

The basic principles will be explained in three introductory examples.

Example 8.4.1:

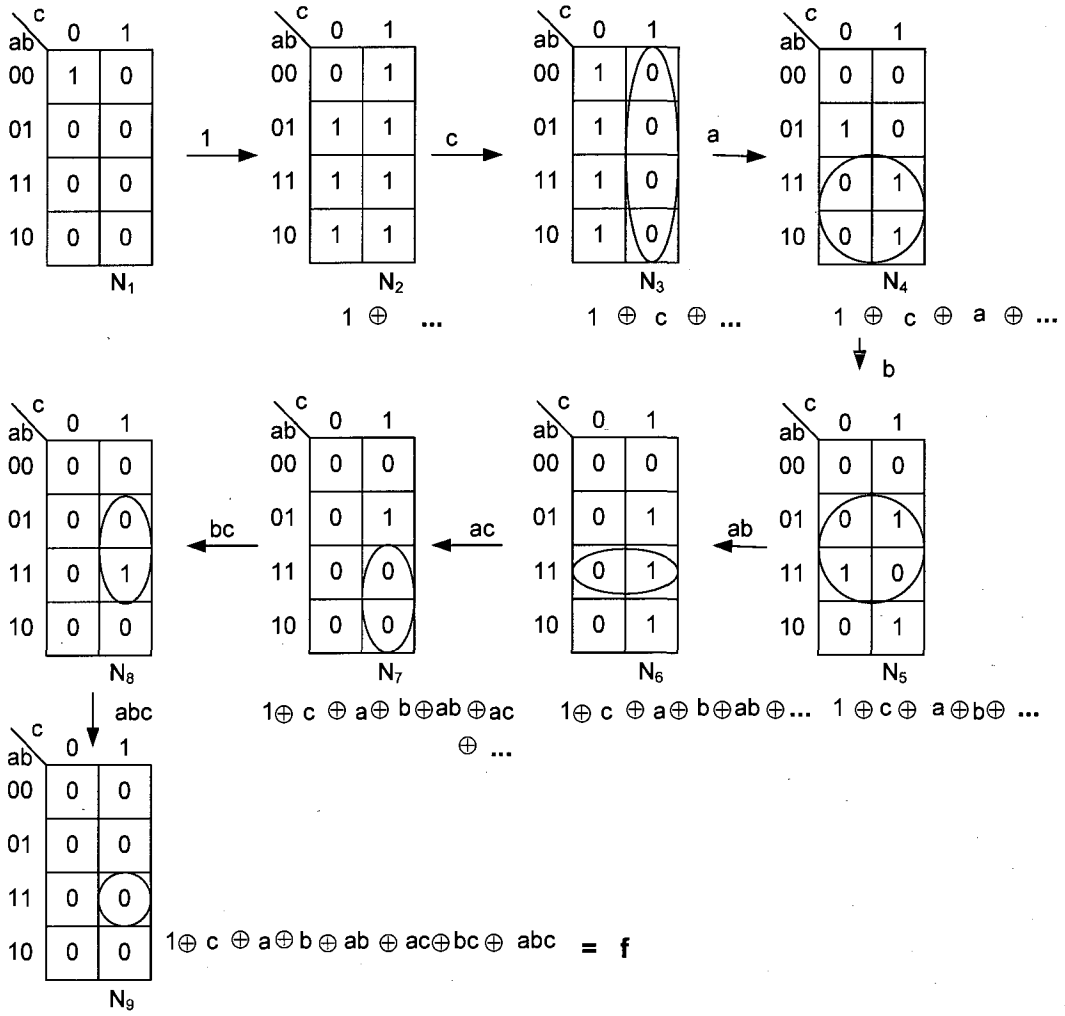


Figure 8.4.1: Minimization of single-output function $f = \overline{a}\,\overline{b}\,\overline{c}$*, assuming the PPRM polarity.*

To demonstrate the counterintuitive nature of the choice of groups for the given polarity let us discuss the minimization of function $f = \overline{a}\,\overline{b}\,\overline{c}$, assuming the PPRM polarity. One branch of the search tree is shown in Figure 8.4.1. The natural first choice would be $\overline{a}\,\overline{b}\,\overline{c}$ but this group is not allowed, as all polarities should be

positive by the PPRM polarity assumption. We see in this example that branching is useless because of symmetry of this function. The solution for PPRM can have a very high cost, as in this example where the PPRM for f = $1 \oplus c \oplus a \oplus b \oplus ab \oplus ac \oplus ab \oplus abc$. However, if the search starts from a good starting point polarity, or if the search uses a good bound of cost, then the algorithm can execute the cut-off early. For instance in this case knowing a literal cost of 3 from solution $\bar{a}\,\bar{b}\,\bar{c}$ would allow to cut-off after reaching node N_5. This example taught us the importance of good starting point, heuristics and bounds in any type of polarity-related minimization such as FPRM or GRM.

Example 8.4.2:

Given is the 2-output function (f_1 (a, b, c), f_2 (a, b, c)) from Figure 8.4.2a.

Assuming polarity [$\bar{a}$, b, $\bar{c}$, $\bar{a}$ b, a$\bar{c}$, b$\bar{c}$, ab$\bar{c}$] the possible product groups for exoring are only 1, $\bar{a}$, b, $\bar{c}$, $\bar{a}$ b, a$\bar{c}$, b$\bar{c}$ and ab$\bar{c}$ and other groups cannot be used according to the assumption of this search method. The partial tree of search is shown in Figures 8.4.2b and 8.4.2c. The nodes of the tree correspond to the remainder functions of [f_1, f_2] after Exoring.

Figure 8.4.2b shows the branch of the solution tree to find first solution – "solution 1". The order of expansions N_i is shown at right of all nodes in Figures 8.4.2b and 8.4.2c. Arrows between nodes are labeled with the selected group symbols and with the costs

of literals cost that correspond to each partially created circuit for the corresponding output function.

Observe that only those product groups allowed for the assumed polarity can be used, thus in the middle branch node N5 (Figure 8.4.2c) where the best choice for minterm 011 is the product group $\overline{a}c$, the algorithm cannot choose it directly as only $b\overline{c}$, $\overline{a}$ b and $a\overline{c}$ are the 2-literal products that can be used. Thus the algorithm has to select the group $\overline{a}$ b for f_1 although this group is of the "equal ones and zeros" type of a group (see node N6). This comes with the selection of good group $b\overline{c}$ and next propagates to Solution 2. After backtrack, the Solution 3 is generated. And after the next backtrack the Solution 4 is generated. We did not discuss expansion $\overline{a}$ bf, $b\overline{c}$ f and $\overline{c}$ f from node N_2 as they have low values of quality function. This example shows how the algorithm can withdraw from bad choices by using the backtracking. Other types of withdraw is by the change of polarity on a higher level of search, implemented by evolutionary methods.

Observe also, that in this example we deal with three cost functions:

1) one-to-zero ratio (heuristic quality function),
2) number of literals (more accurate quality function),
3) number of 2×2 gates (final cost function to be minimized).

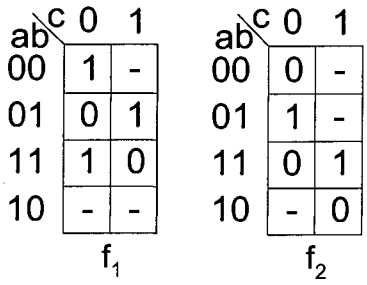


Figure 8.4.2: (a) The 2-output function (f_1 (a, b, c), f_2 (a, b, c)) used in Examples 8.4.2, 8.4.3 and section 8.4.2.

Example 8.4.3:

We use the same 2-output function as in Example 8.4.2.

In this example we assume that the minterms are represented in the ON/OFF set in the Comparison Table (Table 8.4.2.1). The columns of the Comparison Table correspond to polarity coefficients, thus the table from Table 8.4.2.1 corresponds to the polarity (1, ($\overline{a}$), (b), ($\overline{c}$), (a, b), (a, c), (b, $\overline{c}$), (a, b, $\overline{c}$)). Comparison Table is built by the algorithm for each polarity that it reaches. The basis functions of the given polarity (called also standard trivial functions, cubes, or coefficient functions) correspond to the columns of the table. The care minterms in column **abc** correspond to the rows in the table (See Table 8.4.2.1). The second column has all care minters as headers of rows. These minterms may be negated multiple number of times during the synthesis as the "select a group with more ones than zeros" process is iteratively executed for each GRM polarity. Sections 8.4.2, 8.4.3 and 8.4.4 will explain this search method in full detail.

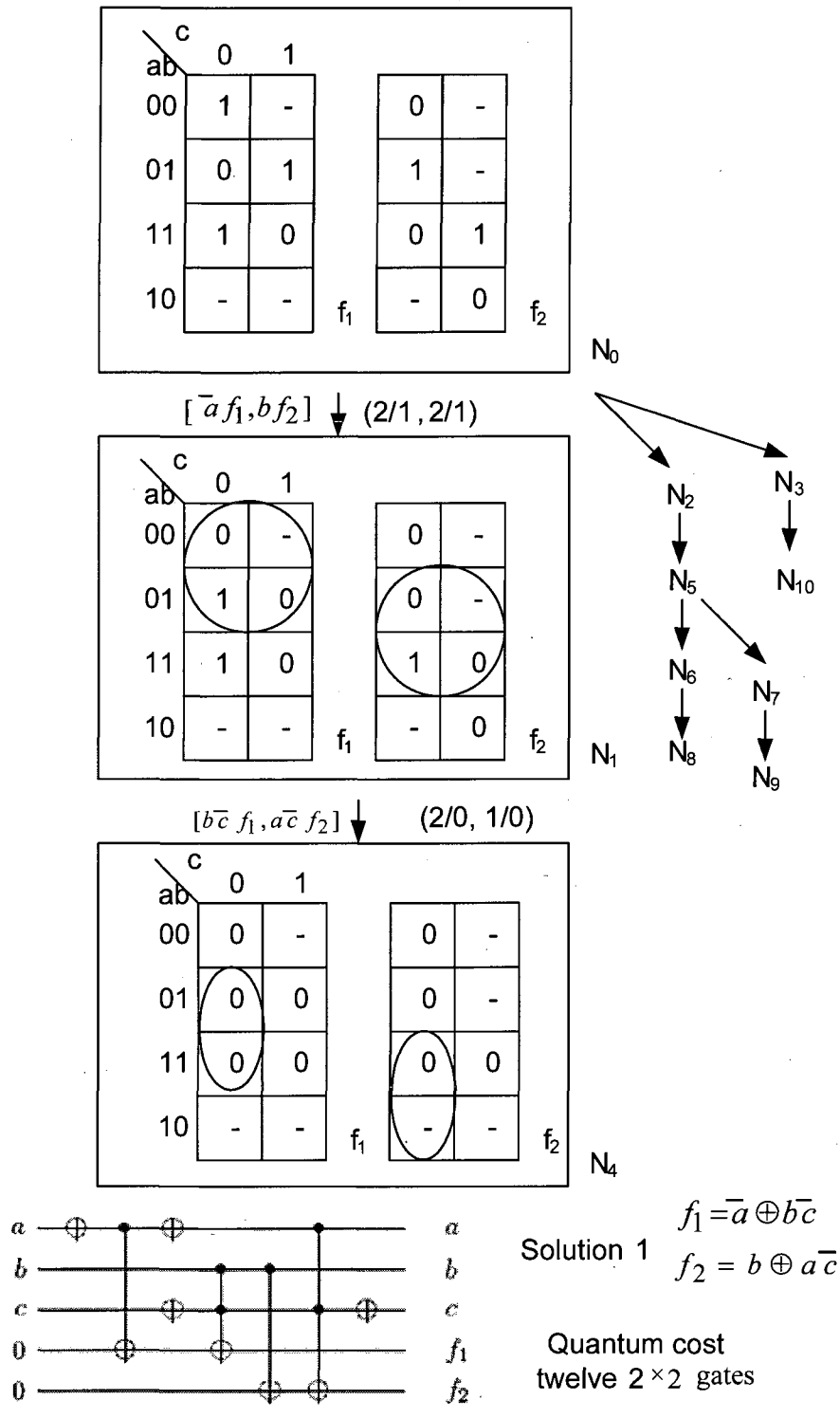


Figure 8.4.2: (b) Partial search tree for 2-output function (f_1, f_2) from Figure 8.4.2a.

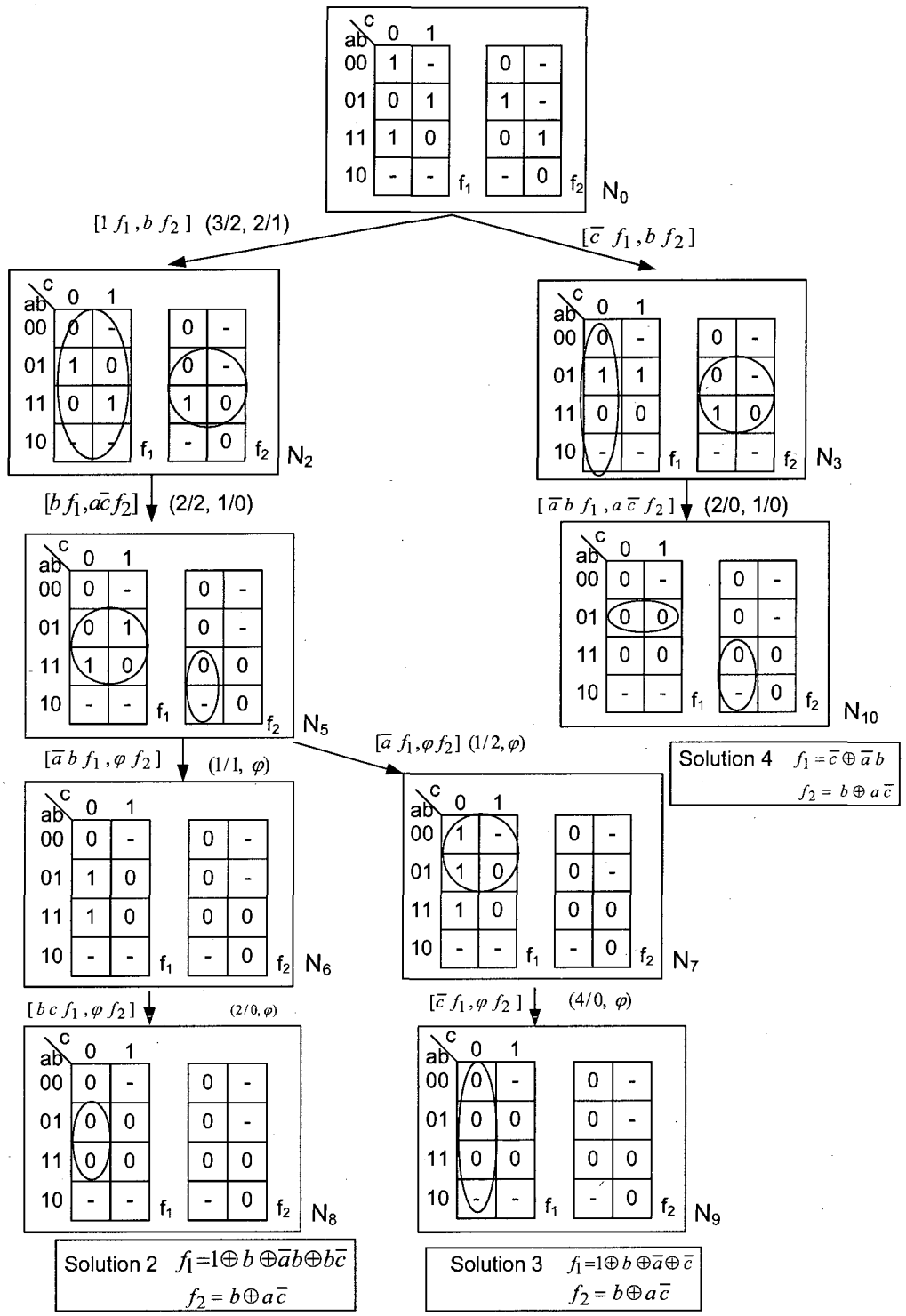


Figure 8.4.2: (c) Partial search tree for function (f_1, f_2) from Figure 8.4.1. These two branches were shown symbolically.

8.4.2. Detailed description of building the Table (Table 8.4.2.1)

To calculate the quasi-minimum GRM form for a given polarity, the ECPS-GRM minimization algorithm creates a table like one in Table 8.4.2.1 with all GRM coefficients for this polarity as columns and all ON/OFF minterms as rows. This table is build for any GRM polarity found by the polarity searching genetic algorithm. The cubes in various output functions of the specification are repeated as separate rows, one for each function (this is illustrated for [f_1, f_2] from example 8.4.3 as in Table 8.4.2.1). The set of the selected columns represents the EXOR realization of all the product terms for the output functions.

The table uses the concept of ON-minterms and OFF-minterms. The ON minterms are marked as active by setting the value of flag ON to "1". Minterms with flag ON = "0" are treated as OFF-minterms. Initially the cells of the table are set to "0". Wherever a minterm matches a coefficient, a "1" is set in a cell at the intersection of the minterm's row and the coefficient's column in the table. The coefficient is the product of literals, represented as a cube of "Cube Calculus" formalism. The matching of the minterm and the coefficient indicates a relation in which all literals from the coefficient have the same polarity as their corresponding literals in the minterm. For instance, for our example, coefficient $\bar{b}$ c matches minterm 001X1 = $\bar{a}$ $\bar{b}$ ce. For all columns that have at least one "1" in some of the rows, the cost is calculated. The column C_{best} with the highest *Column_Cost* is selected. The coefficient cube *Coef*(C_{best})

corresponding to C_{best} is next exored from these output functions f^j, $j=1,...,r$ for which exoring the minterm corresponding to this column with function f^j would bring an improvement in its estimated cost. This is done based on the numbers of "1's" and "0's" in f^j. If there are no better groups available, any matching cube is applied. The cost of the cube selected for the solution is calculated across all output functions f^j.

The *Column_Cost(column$_s$)* is defined as *Column_Cost(column$_s$)* = $\alpha * (N_1 - N_0) + \frac{1}{N_1 + N_0}$, where α is a weighting coefficient and $N_{1/0}$ indicates the number of "1's" or "0's". Note that this formula was chosen as a heuristic means for selecting efficient groups. The $(N_1 - N_0)$ portion assigns a better cost to cubes with many "1's" and few "0's". Whereas following the selection of these groups, the selection of small groups is encouraged with the fractional portion of the *Column_Cost* formula. The α term serves as a balance between these two goals. Herein the even/odd covering problem is heuristically attacked, first selecting the largest groups of 1's, then selecting smaller cubes for the remainder of the terms to be covered. This process aims to iteratively select the best groups, selecting cubes and then successively exoring them with the original function, to create more "0's" (simpler functions) for the remaining necessary cover.

The exoring operation converts some of the ON-minterms to OFF-minterms, and vice versa. This is done by activating and deactivating the flags for each cube (see Table

8.4.2.1). Each ON-minterm covered by a $Coef(C_{best})$ cube in the header of the selected column C_{best} is converted to an OFF-minterm. Each OFF-minterm covered by a selected column is converted to an ON-minterm (flag ON is set to 1). All cubes selected for the output function f^j, $j = 1, \ldots, r$, are triggered for this function. It means that in the first selection, the cube is recorded for this function by triggering the respective bit from 0 to 1. Any next selection of the same cube triggers the respective bit in *Coef_Set*. An even number of selections means no selection and an odd number of selections means a single selection.

For every new selected product group, the contents of the table's cells are modified accordingly. The procedure is repeated until no more ON-minterms remain in the table. The cost of the *solution_Coef_Set* is calculated incrementally with the selection of new product groups.

Observe that the "more-ones-than-zeros" heuristics is only a general principle. This heuristic allows to create various rules to choose "best" groups within search strategies. For instance the opening of a node can be done using several methods:

1. with all groups,
2. with only those groups that satisfy the "more-ones-than-zeros" rule,
3. with only those groups that satisfy the "more-or-equal-ones-than-zeros".

F1	minterms								
ON/OFF	abc	1	a'	b	c'	ab	ac	bc'	abc'
1→0	000	1	1		1				
1→0	011	1	1	1					
1→0→1→0	110	1		1	1	1		1	1
0→1→0	010		1	1	1			1	
0	111			1		1	1		
Iterations 1-3:									
Cost-1		1.20	1.33	0.25	1.33	0.50	0.00	0.50	2.00
Cost-2		-0.80	1.33	-1.75	-0.66	-1.50	0.00	-1.50	0.00
Cost-3		-2.80	-0.66	-1.75	-0.66	-1.50	0.00	0.50	0.00
Cost-4		-2.80	-2.66	-1.75	-0.66	0.50	0.00	0.50	2.00

F2	minterms								
ON/OFF	abc	1	a'	b	c'	ab	ac	bc'	abc'
1→0	010	1	1	1	1			1	
1→0	111	1		1		1	1		
0	000		1		1				
0→1→0	110			1	1	1		1	1
0	101						1		
Iterations 1-3:									
Cost-1		-0.80	0.50	1.33	-0.66	0.50	0.50	0.50	0.00
Cost-2		-2.80	-1.50	-0.66	-0.66	0.50	-1.50	0.50	2.00

Table 8.4.2.1: The Comparison Table illustrating the optimization process for a selected polarity genotype for function [f_1, f_2] from Example 8.4.2 and example 8.4.3.

8.4.3. Iteration Process

8.4.3.1. Selection.

- Select the cube (column) with the highest cost, indicating the best grouping for cube selection. In the case of cubes with equal cost, the selection among these cubes is random. (Note that larger problems often have many choices with equal costs.)
- Include the cube as an EXOR term in the function. (If the cube has been consecutively selected two times (cancelled-out) within the current iteration, then a new cube is randomly selected from the set of all cubes. This allows for the algorithm to jump out of a repetitive selection (loop) of the local maximum within the iteration, and thus continue working towards a solution, while adding some diversity.)

8.4.3.2.Complementation.

In the table (Table 8.4.2.1 in our case), we complement the elements of the ON/OFF set (minterms) that are associated with the selected cube. (These are the ON/OFF set elements in the rows where a "1" exists in the column of the selected cube.) This corresponds to exoring this cube with the data and appending it to the solution.

8.4.3.3.Iteration.

- If all terms (minterms) in the ON/OFF set are "0", then the function construction is completed.
- Otherwise, calculate the new costs and repeat steps 8.4.3.1 - 8.4.3.3.

8.4.4. Repetition and New Polarity Vectors

For a given polarity vector, (i.e. $\overline{a}$, b, $\overline{c}$, ab, ac, b$\overline{c}$, ab$\overline{c}$), n iterations for each function (f_1 and f_2) are conducted. (For this research n = 3.) If the total function cost (number of terms in both functions) has not improved, then a new polarity vector is selected by the GA. The iterative heuristic minimization process from section 8.4.2 is then repeated again for the new polarity. The new polarity can be also selected by a mutation in GA suggested by the results of applying the EXOR logic simplification rules as in Example 8.4.2, presented in Figure 8.4.2.

Although our final version of ECPS-GRM performs more sophisticated search than those from the previous examples, the main principles have been explained in sufficient detail.

8.5. The Detailed description of the ECPS Algorithm Applied to the Approximate Minimization of the Generalized Reed-Muller Form for Incompletely Specified Data

The goal of the presented ECPS-GRM minimization algorithm is to develop a method for minimizing the number of terms, quantum cost or other cost function for the GRM expression. The minimization search space, examining the different polarities, is very large, since for a binary (completely specified) n-variable function there are $n2^{n-1}$ literals and $2^{n2^{(n-1)}}$ polarities.

The incompletely specified function case has the same number of literals and polarities, but the minimization is more difficult and the problem must be viewed differently. This is because for a given polarity of GRM, there exists only one expression (form) for a completely specified function, but many expressions for incompletely specified functions.

The *ECPS-GRM* minimization algorithm performs the GRM minimization; it finds the $Coef\_Set_v$ and $fitness_v$ of the offspring's polarity $Polarity_v$ and stores them together in the GRM-triplet. Details of the minimization algorithm are presented below. New polarities are generated by the GA.

Algorithm 8.5.1: ECPS-GRM (Polarity$_v$, ON/OFF sets of minterms)

1. Create a table with coefficients of polarity as columns and all ON/OFF (minterms) cubes of the multi-output function $\{f_1, ..., f_r\}$ as rows, (repeated for each function in which they stand). Set all cells of the table to zeros.
2. *New_ON_minterms* := ON. *solution_Coef_Set* := 0. *solution_cost(solution_Coef_Set)* := 0. For every new minterm from *New_ON_minterms* mark with a value of "1" in the table every intersection of a column that matches this minterm.
3. For each column C_i that has at least one "1", calculate *Column_Cost*(C_i).
4. Select column C_{best} with the highest value of *Column_Cost*. If several columns have equally high cost, C_{best}, then select randomly from this set of columns.
5. Mark for C_{best} those cubes in output functions f^j (marked functions) that have the highest *Column_Cost*.
6. For each output function f^j that includes a cube marked in step 5 do:

   $f^j := f^j \oplus Coef(C_{best})$

   The exoring creates sets *New_ON_minterms* and *New_OFF_minterms*. Activate

   and modify sets ON and OFF in the table accordingly.
7. Update the *solution_Coef_Set* by triggering the bit of cube C_{best} in the marked output functions f^j of the *solution_Coef_Set*.

min_cost := cost(solution_Coef_Set)

If the same cube is selected consecutively, randomly choose a new cube from the set of all cubes and goto step 3. (The probabilistic selection is done to avoid looping and also creates more diversity in the search.)

8. If there still exist some minterms with a value of "1" in the ON/OFF set goto step 3.
9. Using the ECPS general search mechanism iterate steps 2 to 8 for *n* iterations.
10. Apply Exor logic rules. If they find new polarity then randomly execute P_v : = new polarity resulting from these rules.
11. Return a GRM-triplet: *(Coef_Set$_v$, Polarity$_v$, Fitness$_v$)*.

8.6. Results of Testing on Benchmarks

A test suite was constructed utilizing MCNC benchmark set, completely specified, binary benchmarks. As no incompletely specified benchmarks were readily available, the benchmarks were adapted for these purposes. Using a random selector, 25%, 50%, 75%, and 95% of the benchmark output data was changed to don't cares. These test files are available at www.ee.pdx.edu/polo/function/MCNC_incompletely_specified.

The ECPS-GRM software selects the initial GRM polarity partially from the CGRMIN program run on the completely specified benchmark. (The CGRMIN

program is restrictive, as it minimizes only FPRM logic equations. As the GRM form is less restrictive, the GRM equations should always be reduced to less than or equal the number of terms of an equivalent FPRM equation.) The remainder of the initial polarity vector is specified randomly. The search produces all subsequent GRM polarities through the evolutionary process.

In the iGRMMIN software implementation, a simple Genetic Algorithm was executed, to act on the polarity strings, described as vector strings.

The ECPS-GRM combined program was tested with the benchmark test suite utilizing 25%, 50%, 75%, and 95% don't cares. The results of testing the software, over a test suite of benchmarks are given in Tables 8.6.1 - 8.6.2. The best results for the conducted tests are given and the experimental conditions are noted. Also, when equivalent results were obtained with different population sizes, the test with the shortest run-time is given. The format lists the number of terms after minimization, the generation g_i in which the results were obtained, and the run-time (hours:minutes:seconds). Although better than the previous results from PSU, these results are still worse than those from Sasao and Debnath on few benchmark functions. In contrast to their approach, however, our cost function takes into account also the quantum costs.

Benchmark	Inputs	Outputs	Format: terms, generations, run-time (hours, minutes, seconds)			
			Don't Cares			
			25%	**50%**	**75%**	**95%**
5x01	7	1	6, g2, 00:03:30.22	4, g1, 00:02:44.26	3, g1, 00:00:58.32	2, g1, 00:00:02.01
5x7	7	1	2, g1, 00:03:26.43	2, g1, 00:02:42.29	2, g1, 00:01:05.96	1, g1, 00:00:01.77
bw01	5	1	5, g1, 00:00:10.79	2, g4, 00:00:07.17	1, g1, 00:00:0.91	1, g1, 00:00:0.16
bw19	5	1	2, g1, 00:00:10.13	2, g1, 00:00:07.08	1, g1, 00:00:0.99	1, g1, 00:00:0.14
f21	4	1	2, g1, 00:00:02.97	1, g1, 00:00:00.91	1, g1, 00:00:0.20	1, g1, 00:00:0.07
f56	8	1	2, g1, 00:11:19.65	2, g1, 00:09:21.97	2, g1, 00:05:35.27	2, g1, 00:00:16.90
misex22	6	1	4, g1, 00:00:44.49	4, g2, 00:00:34.44	1, g1, 00:00:01.76	1, g1, 00:00:0.47
misex42	4	1	2, g1, 00:00:02.6	1, g1, 00:00:0.84	1, g1, 00:00:0.14	1, g1, 00:00:0.15
misex56	6	1	3, g1, 00:00:43.29	2, g1, 00:00:30.33	2, g1, 00:00:03.08	1, g1, 00:00:0.49
newcwp	4	5	10, g2, 00:00:11.87	7, g1, 00:00:03.84	3, g1, 00:00:0.48	2, g4, 00:00:0.16
rd53	5	3	25, g5, 00:00:32.21	12, g1, 00:00:17.22	4, g1, 00:00:02.26	2, g3, 00:00:0.35
squar5	5	8	23, g1, 00:01:23.25	21, g2, 00:00:41.62	10, g1, 00:00:06.41	5, g2, 00:00:0.80

Table 8.6.1: Benchmarking on incompletely specified functions with various percents of don't cares. The program is very fast for most test functions.

Comparison for multi-output results are in Table 8.6.2.

Benchmark	Inputs	Outputs	terms, gen., run-time
con175	7	2	10, g1, 00:02:01.56
con195	7	2	4, g1, 00:04.67
rd7375	7	3	23, g1, 00:03:39.83
rd7395	7	3	4, g1, 00:00:07.50
5xp175	7	10	62, g1, 00:08:44.08
5xp195	7	10	11, g1, 00:00:26.75
rd8475	8	4	79, g2, 00:20:30.71
rd8495	8	4	9, g1, 00:00:36.16
log8mod75	8	5	88, g1, 00:30:12.22
log8mod95	8	5	13, g1, 01:08.73
misex195	8	7	15, g1, 00:04:08.0
dc295	8	7	9, g2, 00:00:48.15
clip95	9	5	19, g1, 00:08:40.06
rd84275	8	1	19, g2, 00:06:23.65
rd84295	8	1	3, g1, 00:00:09.24
rd84475	8	1	10, g1, 00:06:12.33
rd84495	8	1	2, g1, 00:00:18.79
9sym95	9	1	2, g1, 00:03:02.78
sao2175	10	1	7, g2, 01:53:55.23
sao2195	10	1	1, g1, 00:15:39.01
misex6475	10	1	1, g1, 01:38:08.4
misex6495	10	1	1, g1, 00:07:56.95

Table 8.6.2: Results for larger and multi-output functions.

8.7. Discussion and Comparison

Few authors [Green91, Mckenzie93, Reige92, Varma91, Zilic95] have considered the problem of PPRM (Positive Polarity Reed-Muller form) minimization for single-output incompletely specified functions. However, with the exception of Zilic and Vranesic [Zilic95], the algorithms are very inefficient for functions that have a large number of don't cares, as the algorithm complexity increases with the amount of unspecified data. Moreover, all these algorithms cannot be adapted to the GRM form, which is quite different from that of the PPRM forms.

For completely specified data, the GRM form has been proven difficult to minimize. The minimization of incompletely specified functions is well known to be more difficult than the minimization of completely specified functions, even for FPRM. For instance, Chang and Falkowski [Chang98] developed a FPRM minimization algorithm for a small percentage of don't cares. In an independent research, Zakrevskij [Zakrevskij95] developed a minimization algorithm for FPRMs that is efficient only for a high percentage of don't cares.

Previous research has shown the GRM Form to be difficult to minimize for the case of completely specified data, both using heuristics [Debnath95, Debnath96] and Genetic Algorithms [Dill98, Dill01]. The iGRMMIN software [Dill01], was the only application of the evolutionary or other methods to minimize GRM forms for

incompletely specified functions. An application combining heuristic search and GAs has not been previously applied to the GRM minimization problem.

It is most difficult to minimize incompletely specified functions with ESOPs that have 5-95% don't cares. It can thus be predicted, for GRMs also, that the minimization of few (<5%) or very many (>95%) don't cares is easier than the case of a medium amount of don't cares. Our results are that program is faster with more don't cares – the higher percent of don't cares, the smaller the processing time. But we do not know and can not know how much quality of results has been sacrificed. Thus to evaluate the quality of our search an exhaustive program should be written which would be very inefficient, as we know from Chapter 7. Therefore it was not done.

8.8. Conclusions

The ECPS-based tool ECPS-GRM has been applied to incomplete GRM minimization and compared to iGRMMIN, the previous algorithm that minimizes incompletely specified data with Generalized Reed-Muller forms. The Generalized Reed-Muller (GRM) forms were selected for this research since they are a good trade-off between cost and high testability [Kalay99]. As much of previous research has presented, the GRM form of AND-EXOR logic has its merits for its high density and testability. This thesis research is the first application of the GRM (a canonical AND-EXOR form) to

the minimization of incompletely specified multi-output functions for the quantum NMR related cost functions.

The software implementation of the ECPS-GRM minimization algorithm was tested over a number of benchmarks. Incompletely specified benchmarks were taken from the *iGRMMIN* data set [Dill01]. Starting with completely specified MCNC benchmarks, a given percentage of outputs were randomly selected for changing to don't cares. These new benchmarks are available from our PSU research group's web site at www.ee.pdx.edu/polo/function/MCNC_incompletely_specified. Minimization test results are given for benchmarks containing 25%, 50%, 75%, and 95% don't cares.

The future extension of this new algorithm to multi-valued logic, using Galois Field algebra [Batisda84, Stewart89], is possible.

Summarizing on background of this work, several concepts have contributed to the results from this chapter. The GRM is a powerful form, because of its canonical, economical (compact logic), and high testability properties. The AND-EXOR logic should be applied to not only completely specified data, but to incompletely specified data as well, which is the more typical case for real-world applications, especially when realizing finite state machines. Together, all of these approaches utilize logic minimization heuristics which are based on human experience. This methodology is

implemented with software. This approach is applicable to traditional computer-automated digital design and synthesis, as well as quantum search (chapters 4, 6, 12 – 15). But, it is also notable that, as this minimization technique is equally applicable for a large number of don't cares (strongly unspecified data) that are characteristic to real-world machine learning problems, it is also applicable to software applications such as Knowledge Discovery/Data Mining and Evolvable Hardware (see chapters 15 and 16).

Observe also that ECPS-GRM can be run after calculating first the best affine preprocessor and its mirror postprocessor, as illustrated in Figure 8.8.1. This is another innovative focus point idea resulting from the overall philosophy of the new approach to quantum arrays presented in this dissertation.

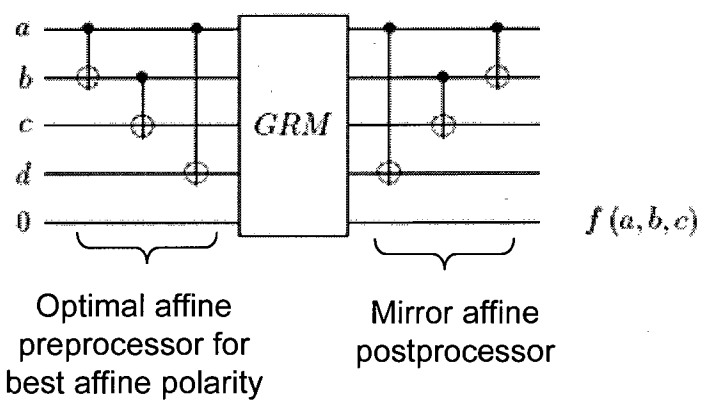


Figure 8.8.1: Illustration of enhancing any GRM synthesis method by using the concept of the affine preprocessor and its mirror postprocessor. The postprocessor is required only in the case when the circuit should be an oracle.

CHAPTER 9

Affine Extensions to Linearly Independent Logic

First part of this chapter is based on literature. Affine extensions and applications of this theory to quantum circuits are new. The introductory material is given for completeness and also to introduce new research results presented in the second part of this chapter.

9.1. Binary ESOP Logic and Affine Extensions

While not as widely utilized for classical integrated circuit design as the AND-OR Sum-of-Product (SOP) logic, the exclusive-or sum-of-product (ESOP) form offers high flexibility paired together with the benefits offered by AND-EXOR logic. This analysis was made, encouraging future design development with ESOP logic, as follows [Song93]:

Functions realized by such circuits can have fewer gates, fewer connections, and take up less area in VLSI and especially, FPGA realizations. They are also easily testable [Fujiwara86, Pradhan87]. It was shown, both theoretically and experimentally [Sasao90c, Sasao91d, Sasao91e, Sarabi92, Salmon89] that ESOPs have on average smaller numbers of terms for both "worst case" and "average" Boolean functions. It was also shown that ESOPs and all their sub-families have their counterparts in logic with multiple-valued inputs: Multiple-valued

Input ESOPs (MIESOPS) [Perkowski89, Sasao94], Multiple-valued Input Generalized Reed-Muller forms [Schaefer91], Multiple-valued Input Kronecker Reed-Muller forms (MIKRMs) [Schaefer93], Multiple-valued Input Generalized Reed-Muller Trees (MIGRMTs) [Perkowski91] and others [Perkowski92]. Logic with multiple-valued inputs (mv logic, for short) generalizes the classical Boolean logic and finds many important applications in logic design [Sasao78, Sasao81, Sasao86, Rudell85]. MIESOPs are never worse than ESOPs, and they were shown to be superior on several classes of functions [Sasao90c, Sasao91d, Sasao91e].

Previously, one of the major drawbacks to utilizing AND-EXOR logic was that function minimization was very difficult. Exact algorithms are intensively time consuming, while heuristic approaches have been limited in both application and quality. All ESOP algorithms for incomplete functions are weak. With the development of EXORCISM-MV-2, a software package providing "efficient minimization of arbitrary ESOP expressions for multiple-output, multiple-valued input, incompletely specified functions" [Song93], the technology mapping to several quantum libraries was made more practical for functions with very small percent of don't cares. In addition to having a very general form, the ESOP has a two-level circuit implementation, which is easily testable. Functions expressed in ESOP equations usually require fewer gates than those of other AND-EXOR forms and can never require more. It is especially true for multi-output functions. They may have however higher quantum costs.

Example 9.1.1: An example of multi output ESOP realized as a quantum cascade is shown in Figure 9.1.1

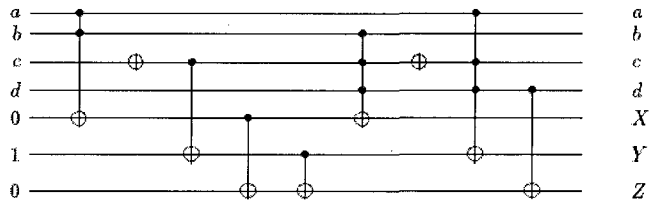


Figure 9.1.1: The quantum array for 3-output ESOP: $X = ab \oplus bcd$, $Y = \bar{c} \oplus cad$, $Z = 1 \oplus ab \oplus d$.

The advantage of ESOP is a total freedom of selecting product groups. This can be however dangerous in terms of cost. As we know only one group with the maximum number of literals is necessary in GRM. As the quantum cost grows exponentially with the number of inputs, these product groups are expensive to be realized in Toffoli gates. On the other hand, the ESOP minimizer can create a very large number of such groups. In chapter 8 we showed how pushing the minimizer to look for specific GRM groups avoided the problem of having many groups with the largest literal costs. Another approach to solve this problem is just to organize the search for the largest (cheapest) groups first, in order to satisfy the "more-ones-than-zeros" heuristics. In our approaches, however, the situation is even better, because we can use not only gates that realize product terms but additionally we can use all kinds of affine gates. This gives a higher probability to find inexpensive groups (not necessarily products) that cover many ones and few zeros. This is simply because now our

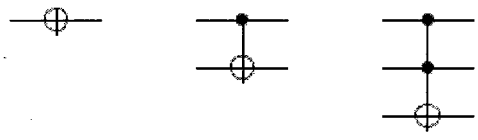
repertoire of patterns of cells to be selected is much larger than for ESOP, FPRM or other similar circuit types.

In Chapter 7, I introduced two new basic ideas to quantum logic synthesis: affine gates and 2-interval symmetric gates. Both these types of gates can be scaled up to many inputs and they do not show an unpleasant characteristics of Toffoli gates that the quantum cost goes quickly up with the number of inputs.

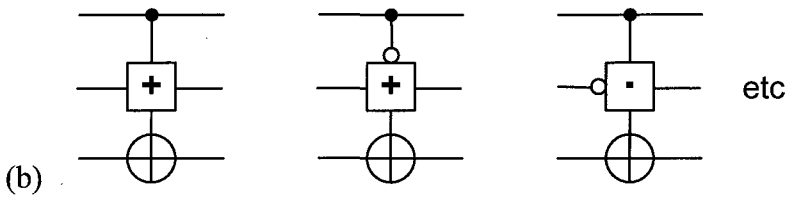
Thus, using the new gates, the design choices for the synthesis algorithms are as shown in Table 9.1.1.

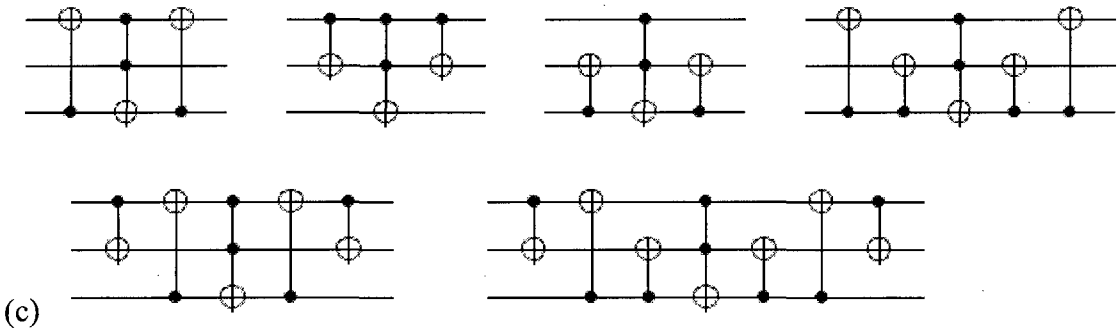
	Traditional	New introduced by this thesis
3×3 functions	NOT, Toffoli, Feynman	NOT, Toffoli, Feynman, Affine Toffoli, 2-input controls Figure 9.1.1.2
4×4 functions		Figure 9.1.1.3
5×5 functions		Figure 9.1.1.4

Table 9.1.1: Comparison of old and new permutative gate libraries.



(a)





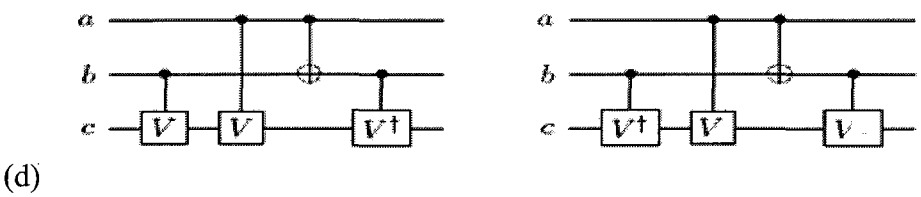
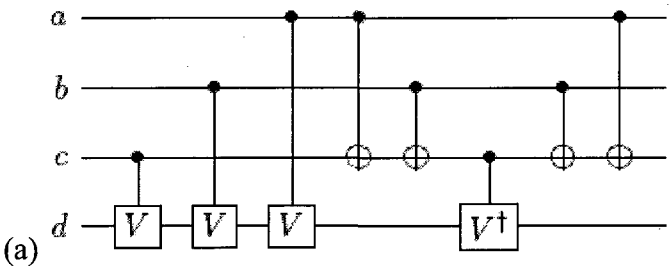


Figure 9.1.2: All gates to be used for synthesis of 3×3 permutative functions. (a) traditional gates, (b) 2-controlled gates, examples, (c) Affine Toffoli gates, examples, they include Fredkin and Miller gates as special cases, (d) Peres family gates, examples.



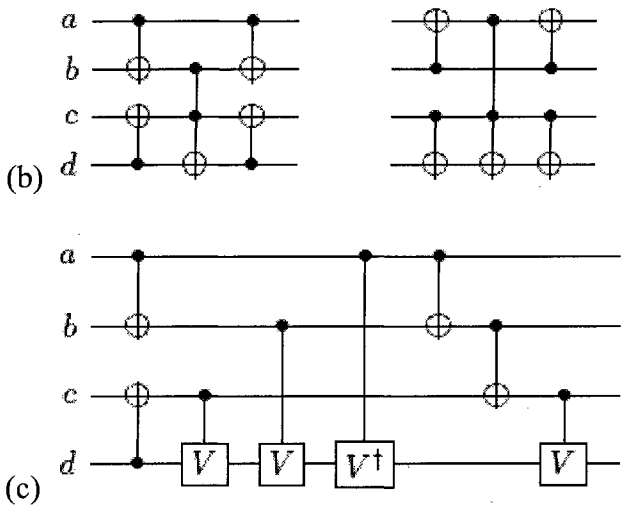


Figure 9.1.3: Examples of all types of 4×4 gates used in our synthesis algorithms, (a) $S^{2,3}$ (a, b, c)⊕d, (b) Affine Toffoli gates, (c) Affine Peres family gates.

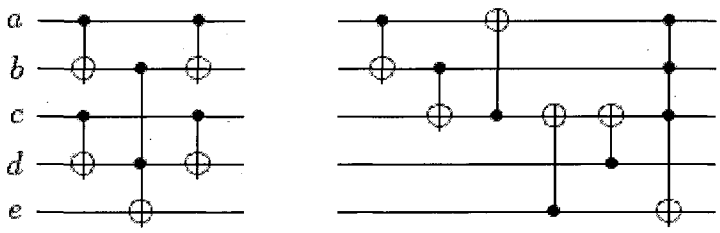


Figure 9.1.4: Some examples of (affine) inexpensive 5×5 gates that are used in our synthesis algorithms.

9.2. Possible approaches of selecting functions to be EXOR-ed

In section 9.1 it was shown that each output in the quantum array can be realized not only as an ESOP but also as an EXOR of certain inexpensive functions. The following possibilities exist:

1. These inexpensive functions are pre-specified as an Linearly Independent Logic family base functions. Examples are base functions of PPRM, FPRM or GRM.
2. These functions are pre-specified but they are not sets of base functions but union of sets of all base functions. For instance, the set of all GRM literal products as found in section 8.1 is the unions of all sets of GRM base functions. The problem of selecting the best groups becomes thus more difficult.
3. These functions not considered now as base functions. They are arbitrary and are selected dynamically to minimize the number of patterns in AND/EXOR decomposition.

All these methods are related on one hand to the material presented in chapters 7 and 8 and on the other hand to the concepts of the Linearly Independent Logic. We have first to review some minimal background of LI logic. LI logic can be discussed in relation to decision diagrams and matrices. We will cover both approaches.

9.3. Linearly Independent Zhegalkin Logic

Reed-Muller Logic Theory was expanded with the introduction of the Generalized Kronecker Expansions to the Zhegalkin Hierarchy [Perkowski97a, Perkowski97c].

For instance, the Zhegalkin Kronecker Reed-Muller Form is one of items from this hierarchy.

Zhegalkin Expansions are linearly independent, AND/EXOR canonical forms. The Zhegalkin Kronecker Reed-Muller Form is obtained when a single expansion from the set of all possible Zhegalkin Expansions is applied in every level of the expansion tree (a variable). Additionally, *"the GRM expansion with functional coefficients is a special case of the LI (Linearly Independent) Expansion with functional coefficients"* [Perkowski97b]. Thus a method was presented in chapter 8 such that an expansion can be determined, enabling a valid GRM with linear independence to be found, given its defining logic table. With this understanding, the Reed-Muller Logic Hierarchy can be related to the Zhegalkin Hierarchy, as described by forms, trees, and decision diagrams [Perkowski97a, Perkowski97c]. The Zhegalkin Hierarchy is a subset of the Linearly Independent Logic Hierarchy [Perkowski97b]. The linearly independent logic and the LI hierarchy include the Reed-Muller Hierarchy of Green and Sasao and all other known and future AND/EXOR forms. It is a very general and powerful approach and our chapter here only scratches the surface of the problem. But we give the first applications of LI to quantum circuit synthesis.

In this section, first the relations between the Reed-Muller, Zhegalkin, and Linearly Independent Hierarchies will be addressed. An example will be given to present a method to compute a multi-variable GRM Expansion in terms of LI theory. As this is

an important component in Zhegalkin Logic, in this manner, Zhegalkin Logic is first informally introduced. Following which, a formal presentation of the Zhegalkin Logic Family is given, with an example. As was previously alluded to, the Reed-Muller Logic Hierarchy can be related to the Zhegalkin and Linearly Independent Hierarchies by demonstrating that all expansions from the set of expansions for a given function are linearly independent. Since the GRM Form in the RM Hierarchy is most central to this research, the analogous form in the Zhegalkin Hierarchy will be here examined.

A method to compute these multi-variable GRM Expansions assuming that the coefficients of the variables are sub-functions of the group of the remaining input variables is here given. First, the GRM Expansion is calculated from the given function $f(x_1, \ldots, x_n)$ for a subset of variables $(x_1, \ldots, x_m)$. The sub-functions SF_i are derived from the original function and shown to be linearly independent. This process is described by the following theorem:

Theorem 9.3.1: [Perkowski97b] *"Given is a function* $f(x_1, \ldots, x_m, \ldots, x_n)$ *such that the set of input variables* $\{x_1, \ldots, x_n\}$ *includes properly the set* $\{x_1, \ldots, x_m\}$. *There exists a unique expansion,* $f(x_1,\ldots x_n) = f_0(x_1,\ldots, x_m)SF_0(x_{m+1},\ldots,x_n) \oplus$

$f_1(x_1,\ldots, x_m)SF_1(x_{m+1},\ldots,x_n) \oplus f_3(x_1,\ldots,x_m)SF_3(x_{m+1},\ldots,x_n) \oplus \ldots \oplus$

$f_{2^n-1}(x_1,\ldots, x_m)SF_{2^n-1}(x_{m+1},\ldots,x_n)$ *where functions* f_i *are the given linearly independent (LI) functions of m variables and the coefficient functions (also called the*

"data input functions") SF_L of the remaining input variables, are determined from the coefficient vector,

$$CV = M^{-1} * FV$$

where $FV(x_{m+1}, ..., x_n)$ is a functional vector of all 2^m cofactors of F, with respect to variables from the set $\{x_1, ..., x_m\}$. In general, M is the matrix of 2^m cofactors of F with respect to variables from the set $\{x_1, ..., x_m\}$. Thus, when m=n, the cofactors with respect to variables $x_1, ..., x_m$ become minterms on these variables (and CV is the vector of coefficients for some given canonical form)" [Perkowski97b].

The *"GRM Universal Module"* can be used functionally as a new type of expansion, by selecting a GRM expression from all possible GRM expressions to expand about (in contrast to the conventional Shannon and Davio Expansions from Chapter 3). This technique is illustrated in Example 9.3.1 below.

Example 9.3.1: The KMap in Figure 9.3.1 presents function f(A,B,C,D). We arbitrarily decide to take GRMs of 2 variables. Choosing one GRM Expansion, out of sixteen possible GRM Expansions for two variables provides a unique expansion for f(A,B,C,D).

AB \ CD	00	01	11	10
00	0	1	0	1
01	0	0	1	1
11	1	1	0	0
10	1	0	0	1

Figure 9.3.1: Function of four variables to Example 9.3.1.

Our goal is to calculate the spectral coefficients by using the equation,

$$CV = M^{-1} * FV.$$

First, the functional vector (FV) of cofactors is derived as shown in Figure 9.3.2 from the rows of the given KMap from Figure 9.3.1. The vector FV is as follows:

$$\mathbf{FV} = \begin{bmatrix} f_{A'B'}(C,D) \\ f_{A'B}(C,D) \\ f_{AB'}(C,D) \\ f_{AB}(C,D) \end{bmatrix} = \begin{bmatrix} C \oplus D \\ C \\ C' \\ D' \end{bmatrix}$$

Figure 9.3.2: Developing the Vector FV from the K-map of Figure 9.3.1. The vector of functions on the right represents the cofactors for respective double-variable cofactors $f_{A^i B^j}(C,D)$ from the left. Cofactor $f_{\overline{A}\,\overline{B}}(C,D)$ corresponds to row A = 0, B = 0 of the KMap in Figure 9.3.1., etc.

The matrix M can be determined by selecting a GRM (from the sixteen possible GRM forms) and solving for all values of (A,B). Demonstrating this process step-by-step, let the selected GRM be as follows.

$f(A,B,C,D) = \overline{A}\,\overline{B} \bullet SF_{\overline{A}\,\overline{B}}(C, D) \oplus BSF_B(C,D) \oplus ASF_A(C,D) \oplus SF_1(C,D)$

Substituting A and B values to the above equation for AB = (0,0), (0,1), (1,0), and (1,1) results in four equations. These are:

A=0, B=0: $f = 1 * SF_{\overline{A}\,\overline{B}}(C, D) \oplus 0 * SF_B(C, D) \oplus 0 * SF_A(C, D) \oplus 1 * SF_1(C, D)$

A=0, B=1: $f = 0 * SF_{\overline{A}\,\overline{B}}(C, D) \oplus 1 * SF_B(C, D) \oplus 0 * SF_A(C, D) \oplus 1 * SF_1(C, D)$

A=1, B=0: $f = 0 * SF_{\overline{A}\,\overline{B}}(C, D) \oplus 0 * SF_B(C, D) \oplus 1 * SF_A(C, D) \oplus 1 * SF_1(C, D)$

A=1, B=1: $f = 0 * SF_{\overline{A}\,\overline{B}}(C, D) \oplus 1 * SF_B(C, D) \oplus 1 * SF_A(C, D) \oplus 1 * SF_1(C, D)$

Equations 9.3.1

We see that the coefficients of the above four equations 9.3.1 can be rewritten to a non-singular matrix M that is given in Figure 9.3.3.

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

(a)

$$M * CV = \begin{array}{c} \\ A'B' \\ A'B \\ AB' \\ AB \end{array} \begin{array}{c} \begin{array}{cccc} A'B' & B & A & 1 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array} * \begin{bmatrix} SF_{\overline{A}\,\overline{B}}(C,D) \\ SF_B(C,D) \\ SF_A(C,D) \\ SF_1(C,D) \end{bmatrix}$$

(b)

Figure 9.3.3: (a) Matrix M, (b) The matrix equation for Figure 9.3.1. The Rows of matrix M correspond to minterms and the columns correspond to the base functions $\overline{A}\,\overline{B}$ *, A and 1.*

Using matrix algebra notation we obtain M * CV = FV , thus M^{-1} FV = CV. This leads to the matrix equation from Figure 9.3.4.

$$M^{-1}FV = CV = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} f_{A'B'}(C,D) \\ f_{A'B}(C,D) \\ f_{AB'}(C,D) \\ f_{AB}(C,D) \end{bmatrix} = \begin{bmatrix} SF_{\overline{A\,B}}(C,D) \\ SF_{B}(C,D) \\ SF_{A}(C,D) \\ SF_{1}(C,D) \end{bmatrix}$$

Figure 9.3.4: Matrix equation using the inverse matrix M^{-1} where M^{-1} FV = CV is the vector of spectral coefficient functions.

Our main equation is thus now given in Figure 9.3.5.

$$\begin{bmatrix} SF_{A'B'}(C,D) \\ SF_{B}(B,D) \\ SF_{A}(A,D) \\ SF_{1}(C,D) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} C \oplus D \\ C \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} C \\ C \oplus D \\ 1 \\ D \end{bmatrix}$$

Figure 9.3.5: Calculation of spectral coefficients. In general, the base functions on variables A and B are of arbitrary type, and the linear combinations of cofactors on variables C and D are also of arbitrary type. Thus LI extends from AND/EXOR logic to arbitrary operators.

To verify matrix M^{-1}, it must be that M * M^{-1} is a unity matrix. This is demonstrated in Figure 9.3.6.

$$M \quad M^{-1}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1+1=0 & 1+1=0 & 1+1=0 \\ 0 & 1 & 1+1=0 & 1+1=0 \\ 0 & 1+1=0 & 1 & 1+1=0 \\ 0 & 1+1=0 & 1+1=0 & 1+1+1=1 \end{bmatrix}$$

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

Figure 9.3.6: Verification of matrix equation for matrices M and M^1.

Now substituting the data into the previous function in the GRM Form,

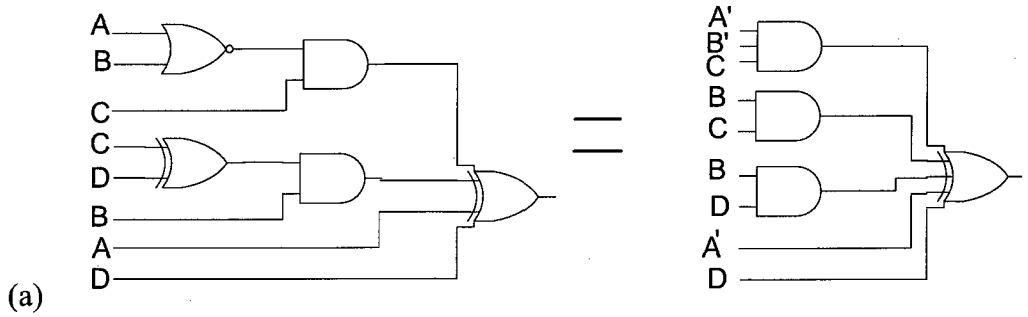
$$f(A,B,C,D) = \overline{A}\,\overline{B} \bullet SF_{\overline{A}\,\overline{B}}(C, D) \oplus B * SF_B(C, D) \oplus$$

$$A * SF_A(C, D) \oplus 1 * SF_1(C, D)$$

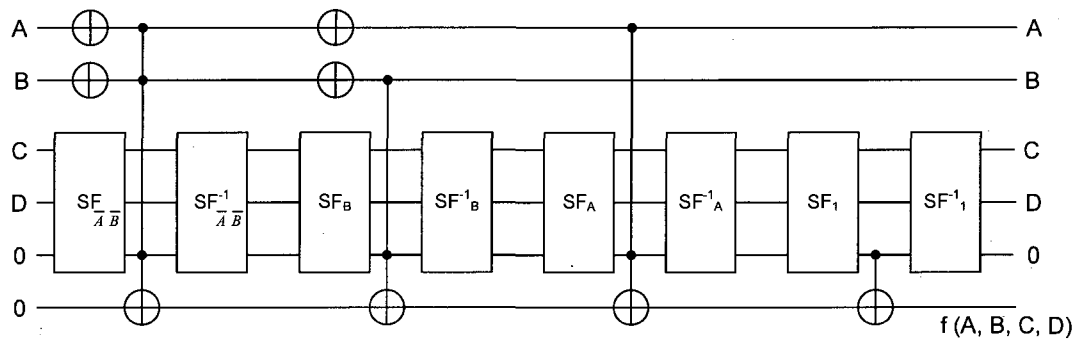
$$= \overline{A}\,\overline{B}\,(C) \oplus B(C \oplus \overline{D}) \oplus A(1) \oplus 1(D)$$

$$= \overline{A}\,\overline{B}\,C \oplus BC \oplus B\,\overline{D} \oplus A \oplus D$$

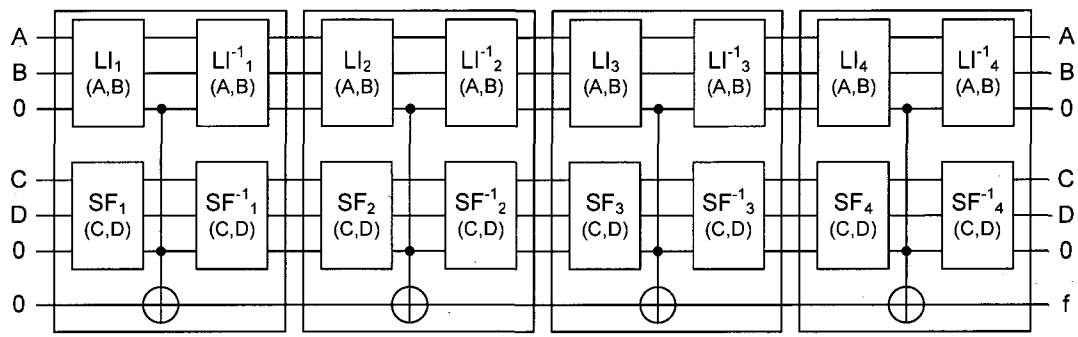
This solution expression corresponds directly to the circuit shown in Figure 9.3.7a. The circuit on the left of this figure comes directly from the above expansion and the circuit on the right is obtained from this first circuit using the flattening operation (X $\oplus$ Y) Z = XZ $\oplus$ YZ. The general pattern for this kind of LI expansions without flattening is given in Figure 9.3.7b. An LI pattern that is even more general is given in Figure 9.3.7c.



(a)



(b)



(c)

Figure 9.3.7: Realizations of LI expansions based circuits (a) Flattening of the circuit obtained from LI spectral expansion. Flattening sometimes simplifies the circuit, but not always.(b) The general pattern of LI expansion with base functions $\{\overline{A}\,\overline{B}, A, B, 1\}$, (c) *Another general pattern of LI spectral expansion. Note that M is an arbitrary nonsingular matrix. Note also the regularity of this and previous circuit.*

The quantum array for the circuit from Figure 9.3.7a before flattening is shown in Figure 9.3.8.

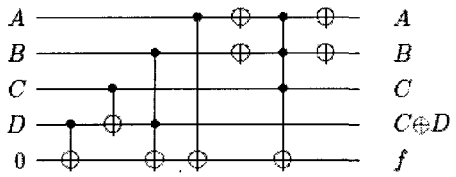


Figure 9.3.8: The quantum array directly corresponds to the circuit from the left part of Figure 9.3.7. No mirror circuit is created here to restore input D and the circuit has only one ancila bit. If one wants to use this circuit as an oracle with input variables A, B, C, D, the mirror circuit to restore D must be added as in previous examples.

Let us observe now few very important facts:

1. The basis functions in the Linearly Independent Logic are not only products of literals as in Reed-Muller Logic but arbitrary linearly independent Boolean basis functions.
2. The basis functions include thus functions and component operators such as a + b, a + $\overline{b}$, a $\oplus$ b, a • b, their combinations and other functions presented in Chapters 3, 4, 7 and 8.
3. All kinds of new functions (gates) that are introduced in this thesis because of their low quantum cost can be included to sets of basis functions of LI expansions. This explains the enormously high power of LI logic in quantum array synthesis that is only partially investigated in my dissertation.
4. Basis functions can be created dynamically for a given function or created and pre-specified once for all for the synthesis algorithms.

5. Unions of sets of basis functions are also useful in synthesis, but using them makes choices of groups more difficult.

A brief description of the Zhegalkin Logic definitions and hierarchy is next given. We will define concepts useful to create sets of base functions.

1. First observe that certain decomposition of an arbitrary function is possible in which, similarly to the Ashenhurst-Curtis Decomposition, the set of all input variables is partitioned into several, disjoint and non-empty subsets, such that the union of all these subsets equals the initial set of variables.
2. Now if the subset of variables from this decomposition has only a single variable then the Shannon Expansion, the Positive Davio Expansion, or the Negative Davio Expansion can be applied to the function for that variable. This creates a standard expansion node as in Chapters 3 and 4, with two edges going out.
3. If the set of variables has more than one variable the expansion node is called the multi-variable node and then the GRM Expansion of certain polarity is applied to this node. It is called the block expansion.

Definition 9.3.1: The Zhegalkin Single Polarity Reed-Muller Form is obtained when the expansion is an arbitrary Zhegalkin Expansion (linearly independent, AND/EXOR

canonical form). The expansion must be the same in all levels of the tree and the input variables must be ordered.

The Zhegalkin Single Polarity Reed-Muller expansion is a counterpart (a powerful generalization) of the Positive Davio, Negative Davio, and Shannon Expansions.

Definition 9.3.2: The Zhegalkin Kronecker Reed-Muller Form (ZKRM) is obtained when a single expansion, from the set of all possible Zhegalkin Expansions, is applied in every level. Thus, in every level of the tree the expansion type is the same, but various expansion types can be used on different levels of the tree.

This expansion is a generalization of FPRM expansions.

Definition 9.3.3: The Zhegalkin Pseudo-Kronecker Reed-Muller Form (ZPKRM) is obtained when any subset of expansion types is applied, with any subset of expansion types per level, for ordered variables.

This expansion is a generalization of Pseudo-Kronecker expansions which use Davio expansions.

Definition 9.3.4: The Zhegalkin Free Kronecker Reed-Muller Form (ZFKRM) is obtained when any expansions, from the set of all possible Zhegalkin Expansions, are applied with any ordering of variables. This is also called free order of variables.

This expansion is a generalization of Free-Kronecker expansions which use Davio expansions and free variable orderings in branches.

As an example of the application of decision diagrams from the Zhegalkin Hierarchy, a Generalized Kronecker Decision Diagram is shown in Figure 9.3.9. (This same general method may be applied for all other canonical forms.) In Figure 9.3.9, the first level describes a Shannon Expansion with respect to x_1 and the second level gives an arbitrary GRM expansion with respect to variables x_2 and x_3. The GRM polarities applied to both the expansion components should be the same. The branches of every node are labeled by linearly independent base functions.

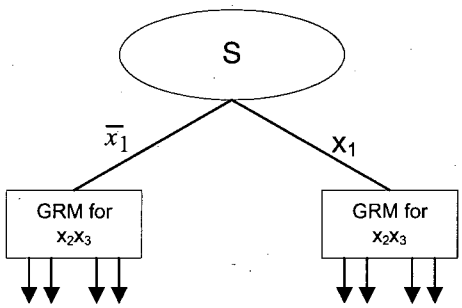


Figure 9.3.9: The principle of mixing single variable expansions (in this case - Shannon applied to variable x_1 on top) and the GRM expansions on bottom.

Example 9.3.2:

Given $f(x_1, x_2, x_3) = \bar{x}_1 \oplus \bar{x}_2 x_3 \oplus \bar{x}_1 x_2 \oplus 1$, perform the expansions shown in the decision diagram in Figure 9.3.9.

To begin, the Shannon Expansion in level 1 must be applied to the given function. Recall that the Shannon Expansion is given as $f = \bar{x} f_0 \oplus x f_1$. First the cofactors for level one are calculated for $f_{x=0}$ and $f_{x=1}$ with respect to x_1.

$f_{x=0} = 1 \oplus \bar{x}_2 x_3 \oplus x_2 \oplus 1 = \bar{x}_2 x_3 \oplus x_2$

$f_{x=1} = 0 \oplus \bar{x}_2 x_3 \oplus 0 \oplus 1 = \bar{x}_2 x_3 \oplus 1$

The decision diagram with the Shannon cofactors shown (in dashed lines) is given in Figure 9.3.10. These are not ordinarily shown in diagrams and are only given here for convenience of explanation.

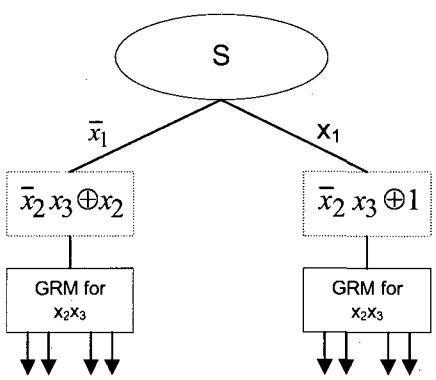


Figure 9.3.10: Calculating of cofactors of x_1 to be further expanded in GRMs.

Next, an arbitrarily selected GRM for the variables x_2 and x_3 is given as $f(x_1, x_2, x_3) = x_2 \oplus \bar{x}_3 \oplus \bar{x}_2\, x_3 \oplus 1$. This GRM is applied for all branches of level two of the decision diagram. This is shown in Figure 9.3.11. Obviously, as in DDs the exhaustive or intelligent search is necessary to find the best decompositions and polarities.

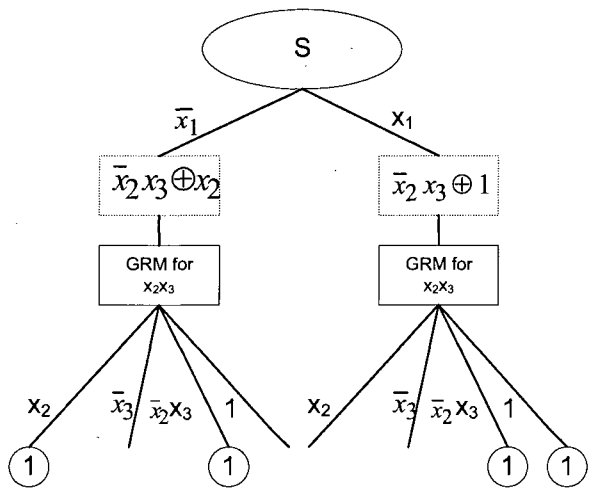


Figure 9.3.11: GRM is applied for all branches of level two of the decision diagram. Note that the unmarked terminals have a coefficient of zero and the dashed boxes are not ordinarily shown.

The algebraic expression can be built from the decision diagram by combining the cofactors and expansion variables in the standard way. Observing Figure 9.3.11, this is done as follows:

$$f(x_1, x_2, x_3) = (x_2 \oplus \bar{x}_2\, x_3)\, \bar{x}_1 \oplus (\bar{x}_2\, x_3 \oplus 1)x_1$$

$$= \bar{x}_1\, x_2 \oplus \bar{x}_1\, \bar{x}_2\, x_3 \oplus x_1\, \bar{x}_2\, x_3 \oplus x_1$$

$$= \bar{x}_1 x_2 \oplus \bar{x}_2 x_3(\bar{x}_1 \oplus x_1) \oplus x_1$$

$$= \bar{x}_1 x_2 \oplus \bar{x}_2 x_3 \oplus x_1$$

$$= \bar{x}_1 x_2 \oplus \bar{x}_2 x_3 \oplus \bar{x}_1 \oplus 1$$

Thus, in this particular case the original given function is re-built.

The corresponding quantum oracle is presented in Figure 9.3.12. Note the use of inverters and the ancilla bit set to 1 initially.

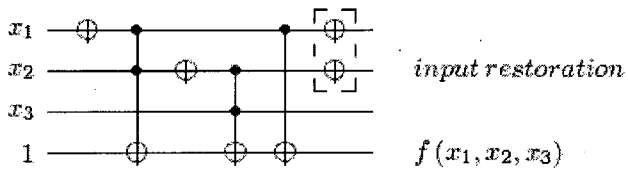


Figure 9.3.12: The final quantum oracle calculated for the function from Example 9.3.2.

Observe that only the simplest concept from Definitions 9.3.1 – 9.3.4 was illustrated in our Example 9.3.2.

The methods presented in this section allow creating multi-level AND-EXOR decomposed structures based on trees and DAGs. They allow also creating families of base functions. The methods of obtaining matrices of base functions efficiently are of our interest in the remaining part of this chapter.

In next sections we will present some families of LI base functions and their corresponding circuits.

9.4. Family of LI base functions using AND and OR operators.

One example of LI circuits are the AND-OR circuits in which the base functions are cascades of AND and OR gates. The family of functions of this type for 3 variables is shown in Figure 9.4.1. Figure 9.4.1a presents symbolically all gates which have either AND or OR gate for every variable, starting with constant 0 on the top. The equation for each gate g_i i = 1, ...8 is written below the gate. Figure 9.4.1b presents the enumeration of minterms. The KMaps corresponding to all g_i functions of certain AND-OR orthogonal family of base functions are given in Figure 9.4.1c. Each minterm present in g_k but absent in g_{k-1} is shown in grey. Equations relating g_i and m_i are presented in Figure 9.4.1d. LI Matrix is given in Figure 9.4.1e.

Based on Linear Independent (LI) equations from Figure 9.4.1d we obtain the following set of linear equations.

$m_7 = g_1$

$m_3 = g_1 \oplus g_2$

$m_5 = g_2 \oplus g_3$

$m_1 = g_3 \oplus g_4$

$m_6 = g_4 \oplus g_5$

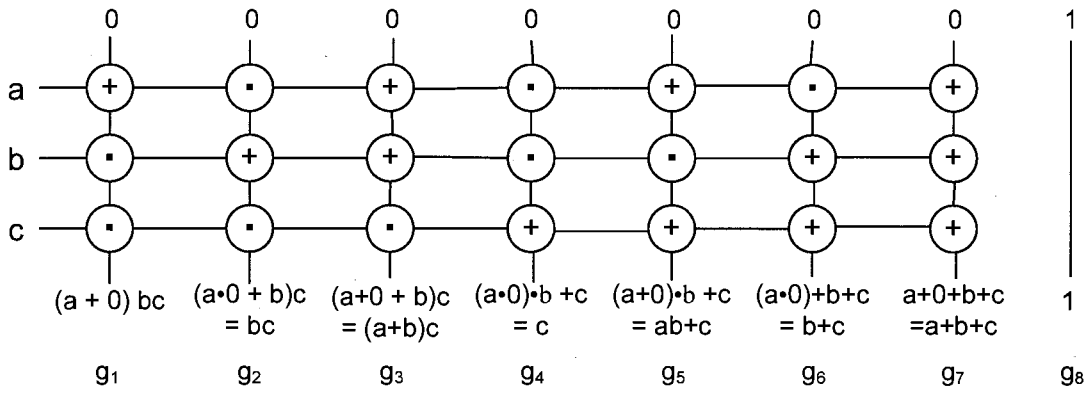
$m_4 = g_6 \oplus g_7$

$m_0 = g_7 \oplus g_8$

These equations can be used to find canonical expansions of every function in form

$$\sum_{o} m_i = \sum_{o} g_i$$

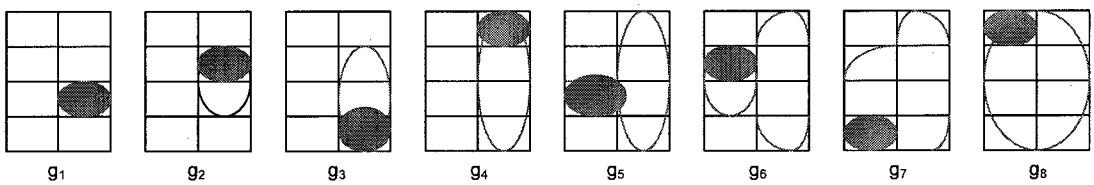
Of course, like in Chapters 7 and 8, we can create affine circuits based on this LI family. An example of a circuit with affine polarity and standard polarity as preprocessors and postprocessor and the AND-OR circuit in the middle between them is presented in Figure 9.4.2.



(a)

ab \ c	0	1
00	m_0	m_1
01	m_2	m_3
11	m_6	m_7
10	m_4	m_5

(b)



(c)

$g_1 = m_7$

$g_2 = m_7 \oplus m_3$

$g_3 = m_7 \oplus m_3 \oplus m_5$

$g_4 = m_7 \oplus m_3 \oplus m_5 \oplus m_1$

$g_5 = m_7 \oplus m_6 \oplus m_5 \oplus m_3 \oplus m_1$

$g_6 = m_7 \oplus m_6 \oplus m_5 \oplus m_3 \oplus m_2 \oplus m_1$

$g_7 = m_7 \oplus m_6 \oplus m_5 \oplus m_4 \oplus m_3 \oplus m_2 \oplus m_1$

$g_8 = m_7 \oplus m_6 \oplus m_5 \oplus m_4 \oplus m_3 \oplus m_2 \oplus m_1 \oplus m_0$

(d)

(e)

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
m_0	0	0	0	0	0	0	0	1
m_1	0	0	0	1	1	1	1	1
m_2	0	0	0	0	0	1	1	1
m_3	0	1	1	1	1	1	1	1
m_4	0	0	0	0	0	0	1	1
m_5	0	0	1	1	1	1	1	1
m_6	0	0	0	0	1	1	1	1
m_7	1	1	1	1	1	1	1	1

Figure 9.4.1: The AND/OR orthogonal family. (a) Schematic diagram of all base functions in AND/OR orthogonal LI family, (b) notation for minterms, (c) KMaps of base functions, new minterms introduced by each successive function g_i are given in grey color, (d) equations for some functions g_i, (e) LI Matrix.

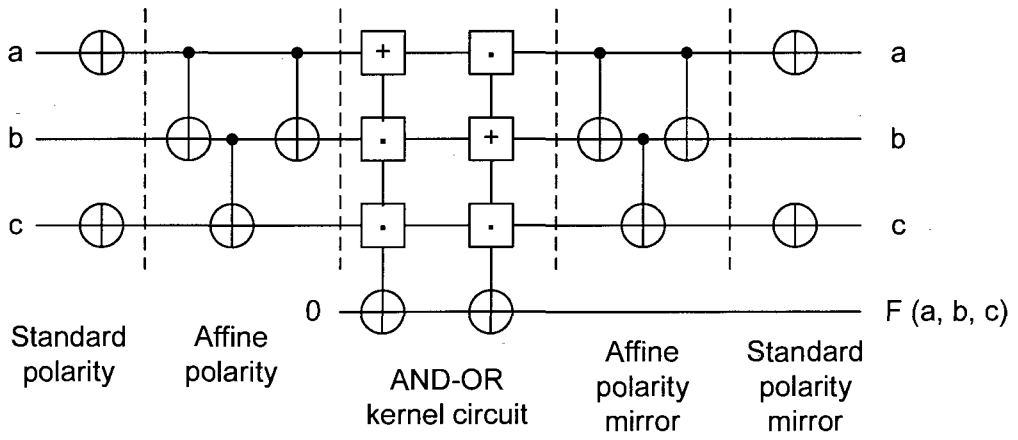


Figure 9.4.2: A general pattern of a complex LI affine circuit that is composed of layers from left to right: the standard polarity layer - a preprocessor, the affine polarity preprocessor, the AND-OR kernel circuit with some subset of base functions from Figure 9.4.1, the affine polarity postprocessor being a mirror of the affine polarity preprocessor and the standard polarity postprocessor being a mirror of the standard polarity preprocessor.

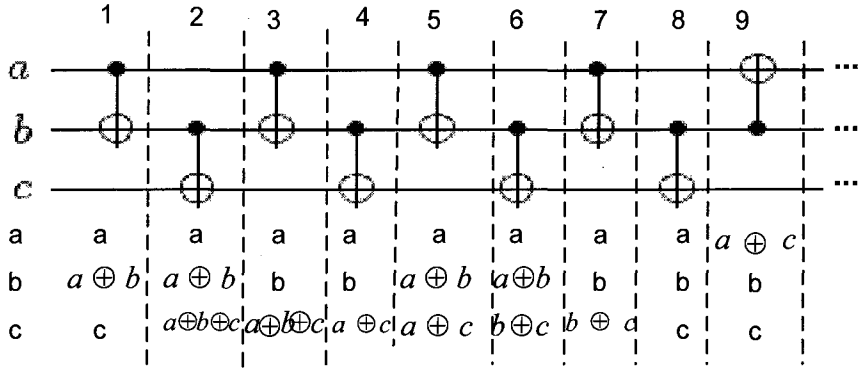


Figure 9.4.3: Part of the pattern for creating all linear combinations of inputs for the affine preprocessor of 3 variables.

It is important in the synthesis of such circuits to be able to generate all 3×3 linear functions for affine preprocessors without ancilla bits. This procedure is illustrated in Figure 9.4.3 and can be a base of the affine polarity generation algorithm. This type of expansions is good to create the interval functions [m_{ij}, ….., m_{ik}] of segments of successive minterms which have applications in cryptography.

Concluding, this section showed another example of LI family of basis functions that can be well realized in a quantum array and finds some useful applications. We showed also how every LI expansion can be enhanced with standard and affine polarities. Combination of these two methods allows to create very efficient quantum arrays, when we know what base family to select or when we can pay time to consider several base families.

9.5. How to Create Inexpensive LI Families?

In the first part of this chapter I introduced LI logic as a logic with linearly-independent (orthogonal) matrices describing families of base functions. Next I showed one illustrative example for several particular families of functions represented as gates and proved that the matrix for each of them is orthogonal. In sections 9.3 and 9.4 I gave more examples of LI families. This constitutes a fundament of creating and using base functions which will be used in this and next sections of this chapter.

The practical question is this:

1. We know some set SI of inexpensive gates on a set of variables and the functions of these gates are not a base family
2. We want to add to SI some set SA of additional gates so that SI $\cup$ SA is certain base family
3. The set SI $\cup$ SA of gates should be not more expensive than the known base families of gates.

This way, we can create new LI families to be used in efficient synthesis algorithms.

Some tricks that we can use to create such sets SI $\cup$ SA are the following:

1) SI is a set of affine gates of all types including the 2-interval gates.
2) SI is built from few gates of a set of root gates other than square-root-of-NOT. For instance gate G = $\sqrt[4]{NOT}$.
3) SI $\cup$ SA is created by performing certain operations on known LI matrices such as matrices of any canonical AND-EXOR logic families (PPRM, FPRM, KRM, GRM, etc) or any other family such as those from previous sections of this chapter.

Some other methods of this type will be illustrated below.

For instance in case of 3-variable functions we know that functions a, b, c, ab, ac, bc, and ab $\oplus$ ac $\oplus$ bc can be realized in expensively using only CV, $CV^{\dagger}$, CNOT base (see Figure 9.5.1.1 and 9.5.1.4).

Unfortunately the set of functions SI(PB) = { 1, a, b, c, ab, ac, bc, g = ab $\oplus$ ac $\oplus$ bc } is not a base family as it does not allow to realize odd functions. Moreover, g is redundant as g = ab $\oplus$ ac $\oplus$ bc. However, using pseudo-base SI(PB) as above (and inverters) we can realize all even functions efficiently. We just need to add the function abc to realize all odd functions. In every odd function we will use however only one gate realizing minterm $a^i b^j c^k$ (Figure 9.5.1.2).

9.5.1. Use of Various Controlled Primitives to create inexpensive gates for set SI.

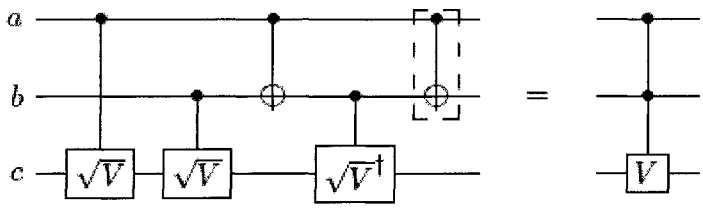


Figure 9.5.1.1: Realization of double-controlled V gate from single-controlled G and $G^\dagger$ gates. This is a fundamental approach to synthesize big Toffoli and Peres gates. Peres family gates are created when the input-restoring circuits from the dotted boxes at right are removed (one gate here).

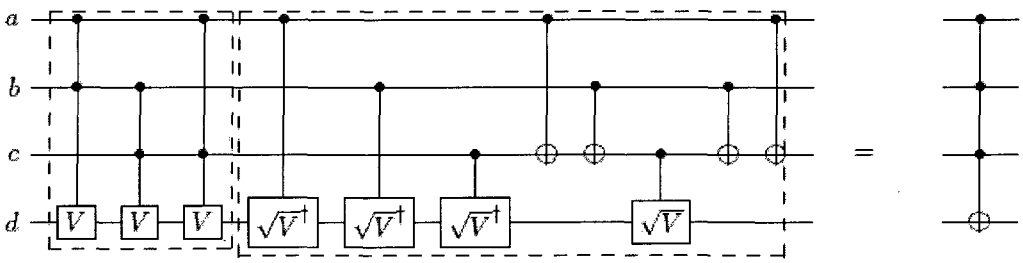


Figure 9.5.1.2: Realization of CCCNOT using double-controlled-V, single controlled G, $G^\dagger$ and CNOT.

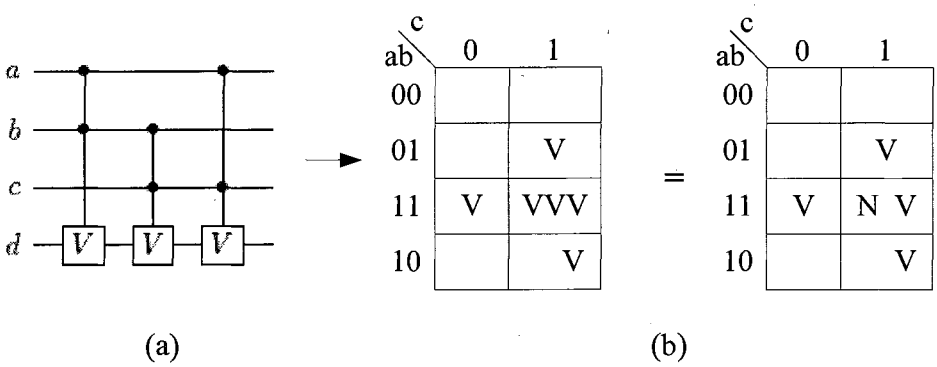


Figure 9.5.1.3: The first auxiliary Circuit (at left in Figure 9.5.1.2) to calculate the 3-controlled Toffoli (a) Circuit, (b) QMap analysis.

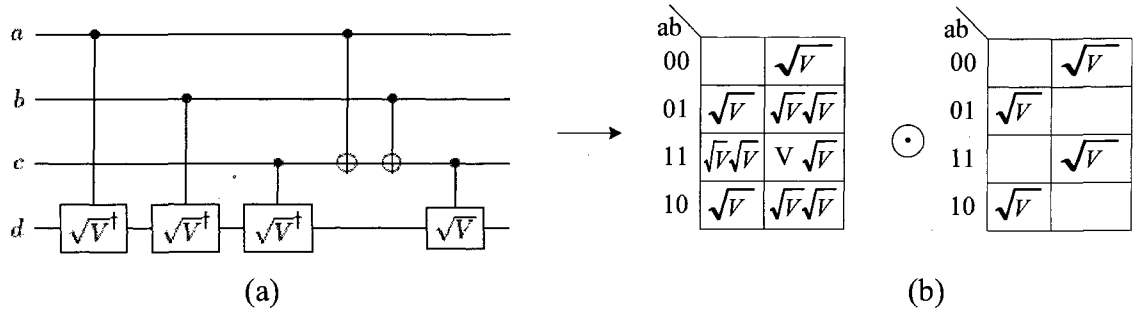


Figure 9.5.1.4: The analysis of the second auxiliary circuit from Figure 9.5.1.2. (a) the circuit, (b) its QMap analysis.

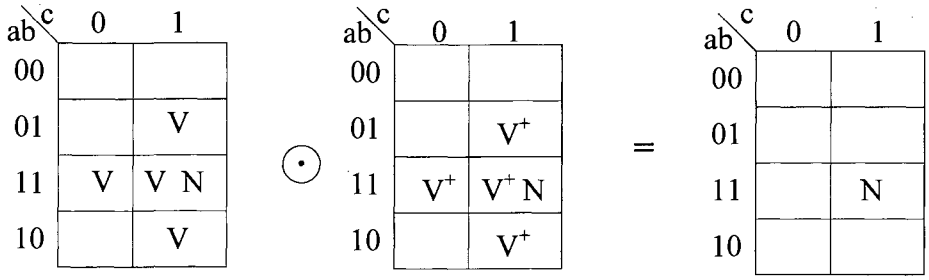


Figure 9.5.1.5: The final QMap analysis of the circuit from Figure 9.5.1.2.

Figure 9.5.1.1 shows how to build the double-controlled-V gate from G and $G^\dagger$ gates. Having now such a gate we can create a triple-controlled Toffoli gate with no ancilla bits (Figure 9.5.1.2). This way, we can create large families of SI base functions.

In my research I created many function candidates for inexpensive SI sets. To create such functions I needed a method to verify my solutions. Examples of using such analysis method for component subfunctions to verify the correctness of our generation method are presented in Figure 9.5.1.3, Figure 9.5.1.4 and Figure 9.5.1.5.

As a result of this generation process, we dispose a 3-controlled Toffoli gate with no ancilla bits to be used in odd functions of a, b, c and potentially in pair functions (Figure 9.5.1.6). The same circuit can be built with the CV-based 3×3 Toffoli gates (Figure 9.5.1.7) which method requires however an ancilla qubit.

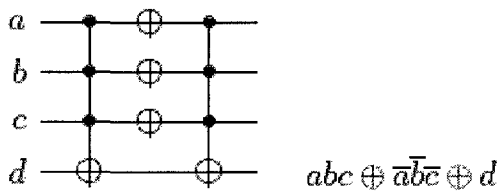


Figure 9.5.1.6: The (inefficient) quantum array for ESOP with 4 × 4 Toffoli gates. This is a minterm pair function of 3 variables.

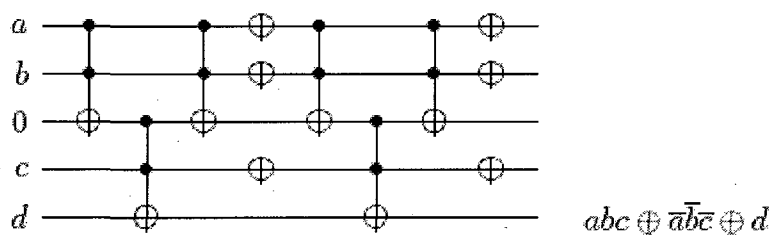


Figure 9.5.1.7: 2-inputs Toffoli for 3 variable ESOP. Realization of the circuit from Figure 9.5.1.6 using 3 × 3 Toffoli. When we replace all 3×3 Toffoli gates with their CV/CV† 2×2 gates we can understand how complex is in reality the quantum sequence of NMR pulses to realize the seemingly "simple" gate from Figure 9.5.1.6.

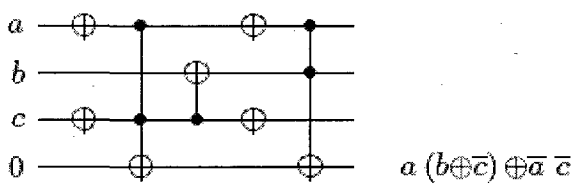


Figure 9.5.1.8: Using factorized GRM for the function of the circuit from Figure 9.5.1.6. This is the least expensive circuit for a "minterm pair" function of 3 variables.

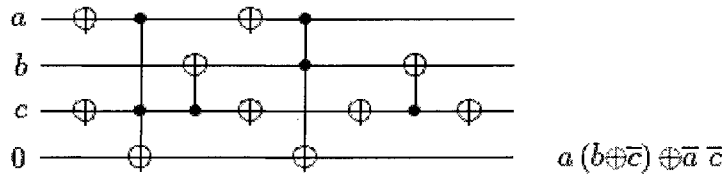


Figure 9.5.1.9: Modification of the circuit from Figure 9.5.1.8 to make it an oracle.

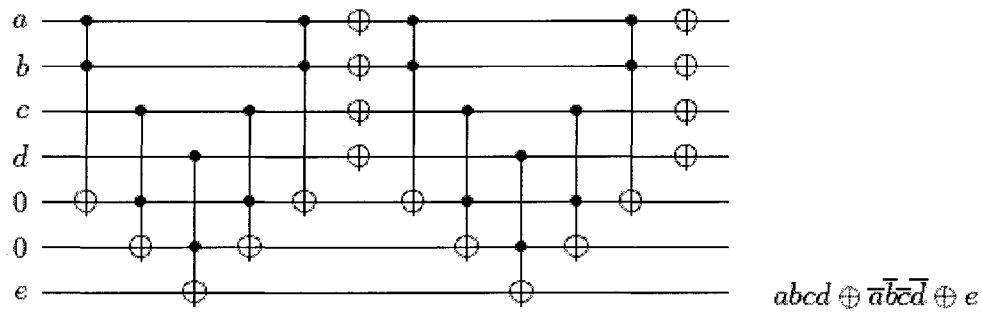


Figure 9.5.1.10: Realization of the "minterm pair" function of 4 variables $abcd \oplus \overline{a}\,\overline{b}\,\overline{c}\,\overline{d} \oplus e$ *using 3 × 3 Toffoli and two ancilla qubits.*

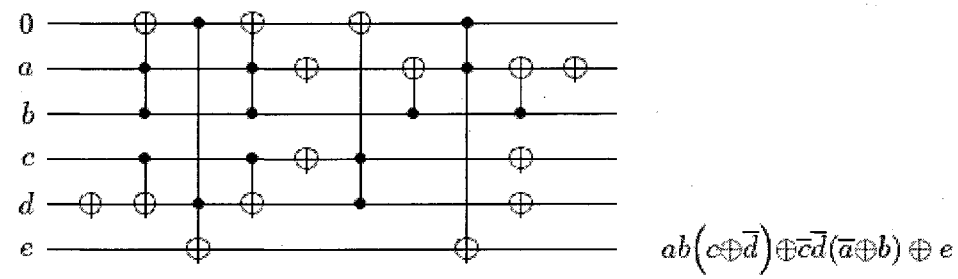


Figure 9.5.1.11: Naïve factorization for the oracle type circuit for $abcd \oplus \overline{a}\,\overline{b}\,\overline{c}\,\overline{d} \oplus e = ab(c \oplus d) \oplus \overline{c}\,\overline{d}\,(\overline{a} \oplus b) \oplus e$ *two ancilla qubits for the "minterm pair" function of 4 variables. This is the least expensive realization of the "minterm pair" function of 4 variables.*

No. of Variables	How Calculated?	To	Fe	N	ANC	Total Cost 2×2 in gates
3	ESOP	6	0	5	2	30
3	GRM	2	2	6	1	11
4	ESOP	10	0	8	3	50
4	GRM	5	4	6	2	29

Table 9.5.1.1: Cost calculations for minterm pair gates for three and four variables : Table for minterm pair functions. To = number of standard Toffoli gates, Fe = number of Feynman gates, N = number of inverters, ANC = number of amcilla bits.

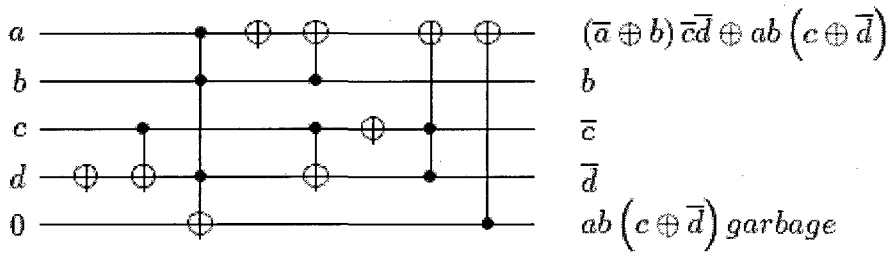


Figure 9.5.1.12: The circuit for $f = ab(c \oplus d) \oplus \overline{c}\,\overline{d}\,(\overline{a} \oplus b)$ *with one ancilla bit which is not designed to be an oracle.*

Using the above techniques I generated several candidates for base LI functions. Some examples of them are given in Figures 9.5.1.7 – 9.5.1.12. It is hard to say if the design of these functions based on CV/CV† or based on CG/CG† 2×2 quantum primitives is better. It depends on technology and functions to be constructed:

1. Observe that every circuit can be rewritten to another function and re-synthesized (Figure 9.5.8) which may again affect the choice of the component gates.

2. The CV and CG gates can be mixed within a single gate (Figure 9.5.1.2).
3. The solution may depend also on the requirements for our circuit: is this circuit supposed to be an oracle or not necessarily an oracle (see Figure 9.5.1.9).
4. Finally, the answer may depend on the size of the gate – Figure 9.5.1.11 has a case of a HD4 pair in a 4 wire space of qubits a, b, c, d – adding one ancilla qubit was necessary.

As every function can be decomposed to dual-cube functions (chapter 7) it is interesting to know the quantum costs of minterm pairs functions versus single minterm functions. Table 9.5.1.1 shows that for functions of 3 and variables synthesis with dual-cube gates leads always to smaller quantum costs. Such functions should be then included to the base function families.

9.5.2. Symmetric Base Functions.

Another topic related to the generation of base functions sets SI or SI $\cup$ SA is the generation of the inexpensive circuits to realize symmetric functions. Figure 9.5.2.1a presents the realization of some symmetric functions. Observe that all single-index symmetric function can be obtained by exoring these inexpensive symmetric functions. For instance $S^{1,2} = S^{2,3} \oplus S^{1,3}$ (Figure 9.5.2.1b), $S^2 = S^3 \oplus S^{2,3}$, $S^1 = S^3 \oplus S^{1,3}$.

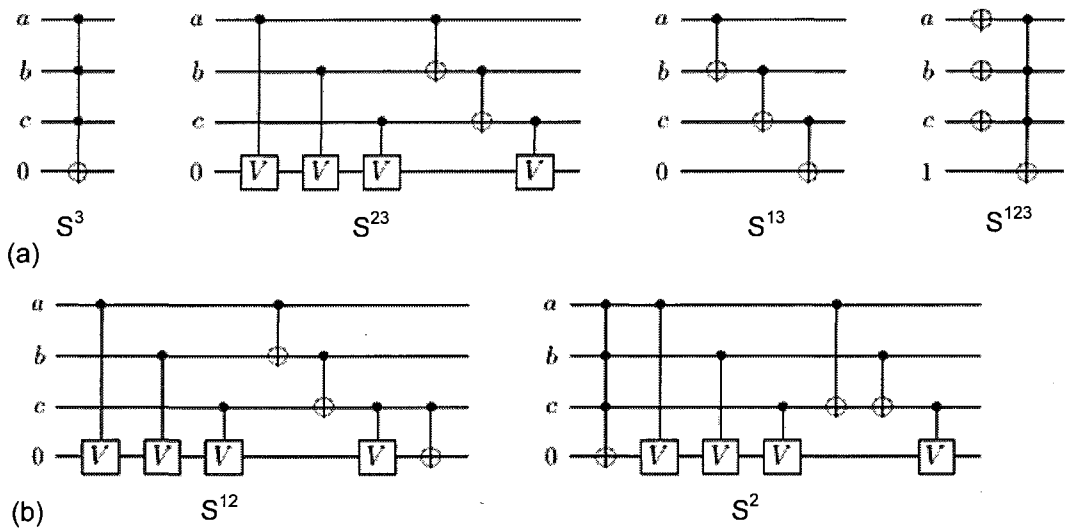


Figure 9.5.2.1: Examples of inexpensive arrays for symmetric functions of three variables with only one ancilla qubit each.

9.5.3. Big Base Functions.

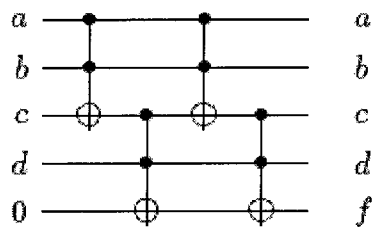
When creating base functions for functions with many variables the typical questions are of the following types:

a) What is the best cost of group *abcd* in the space of four quantum wires?

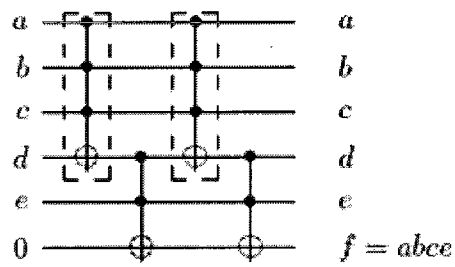
b) What is the best cost of group *abcd* in the space of five wires?

c) What is the best cost of the "minterm pair" functions such as $abcd \oplus \overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$ in the space of four wires? In the space of five wires? Etc.

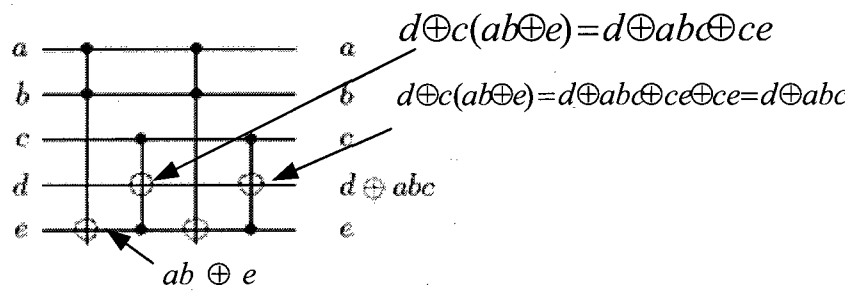
Some design principles to illustrate these questions are given in Figure 9.5.3.1.



(a)



(b)



(c)

Figure 9.5.3.1: Typical tricks to realize large gates. (a) gate f = abd realized in the space of 5 qubits (wires), (b) gate f = abce realized in the space of 6 qubits. Each of the 4×4 Toffoli gates can be realized with G, $G^{\dagger}$ gates and no ancilla or V, $V^{\dagger}$ gates and no ancilla as in Figure 9.5.3.1c. (c) realization of 4×4 Toffoli gate in the space of 5 wires with no ancilla bits.

We can prove the following Theorem.

Theorem 9.5.3.1: The (n – 1) × (n – 1) Toffoli gate can be build in the space of *n* wires.

Proof. See Figure 9.5.3.1.

Based on Theorem 9.5.3.1 every function of n – 1 variables can be realized with n + 1 qubits and every even function of n – 1 variables can be realized with qubits.

Tables similar to Table 9.5.3.1 can be created for the following categories of functions:

a) Odd functions of single minterm for n = 3, 4, 5, etc.

b) 2–interval functions,

c) Symmetric functions.

Such tables help to create libraries of inexpensive gates for re-use as the base functions in LI families.

9.5.4. Creating LI matrices from LI matrices by operating on them.

Rows of a LI matrix represent functions of the LI basis functions family described by this matrix. For instance Figure 9.5.4.1 presents a LI matrix of FPRM with base functions 1, $\bar{a}, \bar{b}$ and $\bar{a}\ \bar{b}$. Because when we exor rows of LI matrix we obtain another

LI matrix, by exoring functions corresponding to rows we obtain a new LI family of (new) base functions. This exoring can be done one at a time, as shown at the right of the matrix in Figure 9.5.4.1 ($\overline{b} \oplus \overline{a}\,\overline{b} = (1 \oplus \overline{a})\overline{b} = a\overline{b}$). The new family of base functions is {1, $\overline{a}, \overline{b}$ and $a\,\overline{b}$}. So we obtain certain GRM expansion from an FPRM expansion, nothing new conceptually, but this is only one example of creating bases. Applying this method to larger matrices in all possible ways we can however create (in theory) any new LI family based on binary logic (for instance Figure 9.5.4.2 creates base functions that do not exist in GRM).

$\overline{a}\,\overline{b}$	$\overline{a}\,b$	$a\,\overline{b}$	$a\,b$		
1	1	1	1	1	1
1	1	0	0	$\overline{a}$	$\overline{a}$
1	0	1	0	$\overline{b}$	$\overline{b}$
1	0	0	0	$\overline{a}\,\overline{b}$	$a\,\overline{b}$

Figure 9.5.4.1: Spectral Matrix with minterms as columns and basis functions as rows – this is a change of basis matrix.

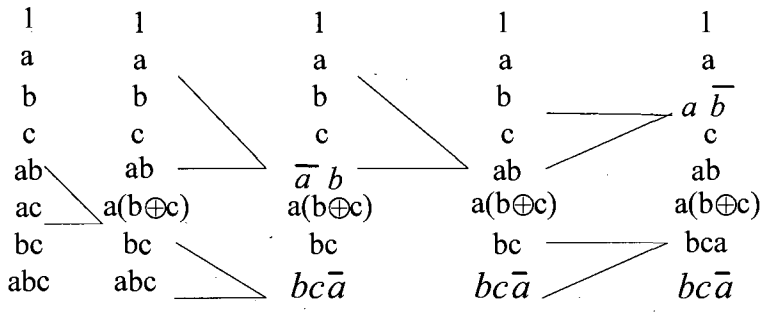


Figure 9.5.4.2: Step-by-Step generation of a sequence of families of Linearly Independent base functions using exoring and starting from PPRM base. Many types of butterfly diagrams and recursive (tree search) algorithms can be adapted to perfom this kind of processing to create new orthogonal bases.

Figure 9.5.4.2 gives example of generating families of base functions. The exoring operations are drawn as arrows from two arguments of the EXOR operator. Thus for instance the new base function a (b $\oplus$ c) is created by Exoring base function ab and ac. This new base function replaces base function ac.

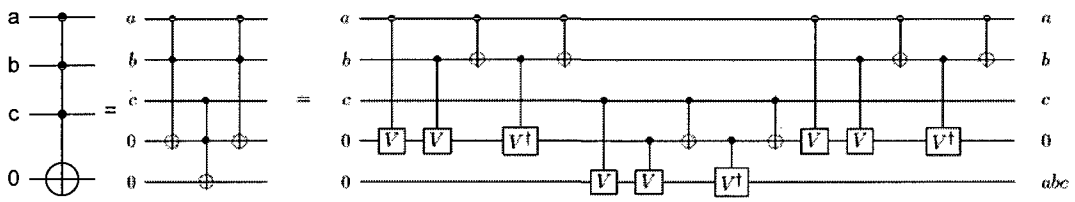


Figure 9.5.4.3: Realization of oracle f = abc with two ancilla bits and 2 × 2 quantum primitives. Observe mirror at right. The increased cost of this recursive expansion can be realized when we look to the right part of this figure. This helps to appreciate the methods to reduce the cost of "big gates".

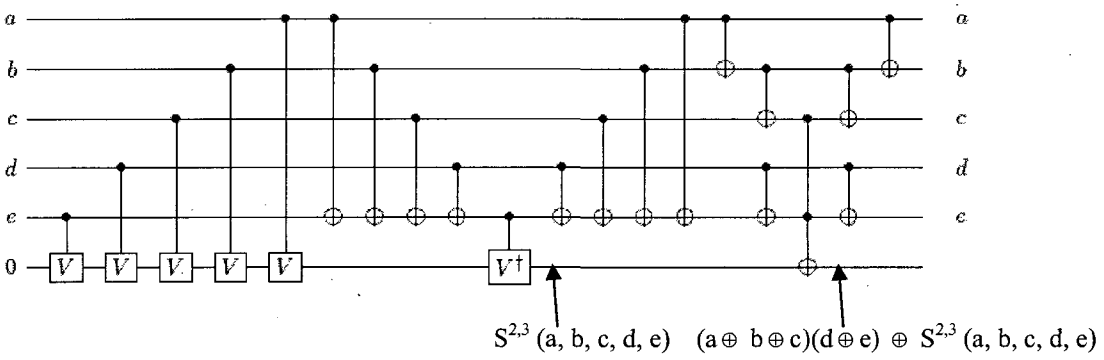


Figure 9.5.4.4: An Oracle for function $S^{2,3}$ (a, b, c, d, e) $\oplus$ (a$\oplus$b$\oplus$c)• (d$\oplus$e). This complex affine circuit is a composition of circuit for the 2 – interval function $S^{2,3}$ (a, b, c, d, e) and the affine Toffoli gate (a$\oplus$b$\oplus$c)• (d$\oplus$e). Observe mirrors. The linear circuit in qubits a, b, c, d, e can be optimized, for instance by removing two dashed CNOT gates.

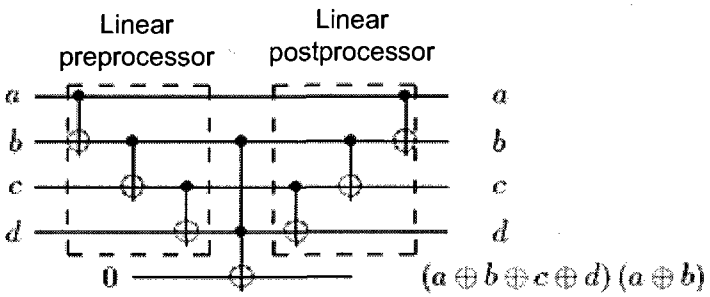
When all base functions are created, we can design and optimize their quantum gates, calculate costs and store in a library. For instance, the cost of f = abc from Figure 9.5.5.3 can be calculated as 15 2×2 quantum primitives. Figure 9.5.4.4 presents an oracle for a more complex function realized with complex affine gates and drawn here to calculate the cost with 2×2 quantum primitives.

9.5.5. Finding All (or some) Affine Functions to Construct Base Functions.

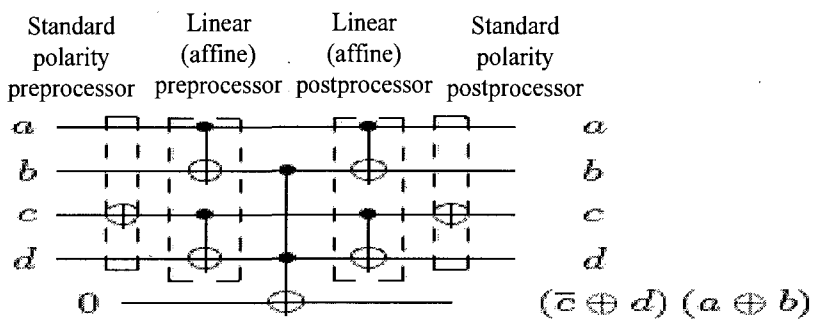
Another issue when realizing affine LI circuits with reduced costs is how to find all affine functions to be used as affine polarities. We will prove that if standard polarities are used as a preprocessor together with a linear preprocessor there is no need to take negations in the affine preprocessor. Similarly no pairs of groups included in one another should be considered for a preprocessor. For instance, (a⊕b⊕c⊕ d) and (a⊕b) are equivalent to and ($\bar{c}$ ⊕ d) and (a⊕b), as proved below.

$(a \oplus b \oplus c \oplus d)(a \oplus b) = a \oplus ab \oplus ba \oplus b \oplus ca \oplus cb \oplus da \oplus db$
$= 1(a \oplus b) \oplus c(a \oplus b) \oplus d(a \oplus b)$
$= (1 \oplus c \oplus d)(a \oplus b)$
$= (\bar{c} \oplus d)(a \oplus b)$

Thus the gate from Figure 9.5.5.1a can be realized as in Figure 9.5.5.1b. And vice-versa.



(a)



(b)

Figure 9.5.5.1: Illustration to general construction methods of affine gates with pre- and post-processing. (a) Realization of non-optimal affine Toffoli gate with {a, b} ⊂ {a, b, c, d}, (b) The optimal circuit replacing the circuit from Figure 9.5.5.1a.

This observation reduces the search to generate all affine functions. We can now avoid repeated generations of the same affine function and reduce the cost of affine gates.

Figure 9.5.5.2 shows how to systematically generate groups of product groups that are not mutually included (part of the tree is shown only). Methods from chapter 6 can be

used to generate such groups. Note that some subsets of correct groups are not usable for Toffoli gate. For instance the node of the tree {(ab, ac, bc)} which corresponds to set of linear functions {(a⊕b) ,(a⊕c), (b⊕c) } is useless because (a⊕b) (a⊕c) (b⊕c) (a⊕ac⊕ab⊕bc) (b⊕c) = 0.

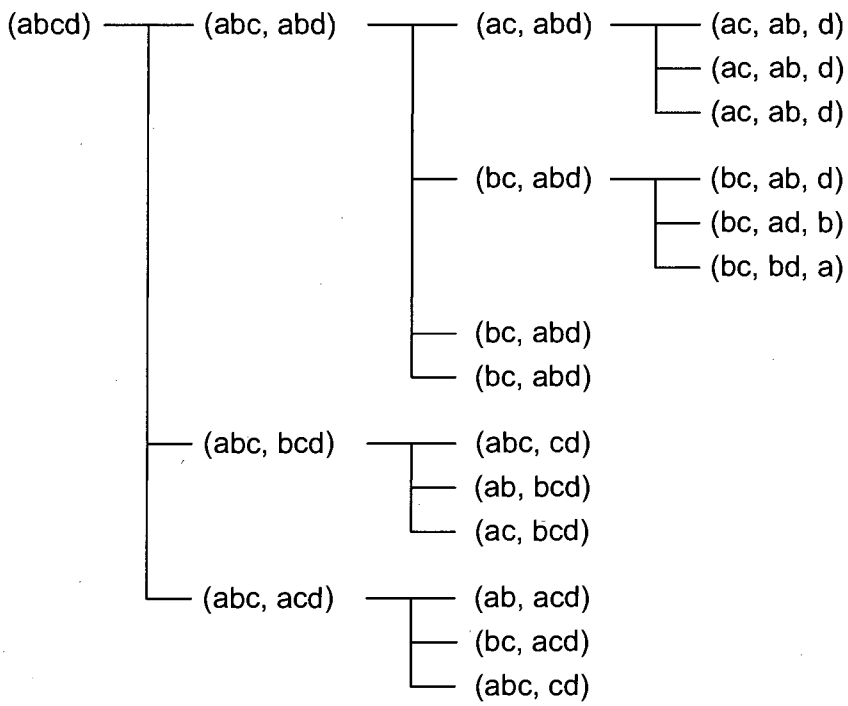


Figure 9.5.5.2: The complete tree method (chapter 6) to create all possible affine preprocessors to gates on four input variables. Each string of variables corresponds to a linear function of these variables, for instance bcd ↔ b⊕ c⊕d.

9.5.6. KRM-Like and Other Mixed Forms.

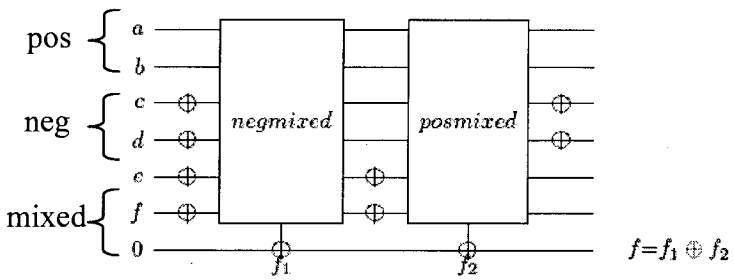


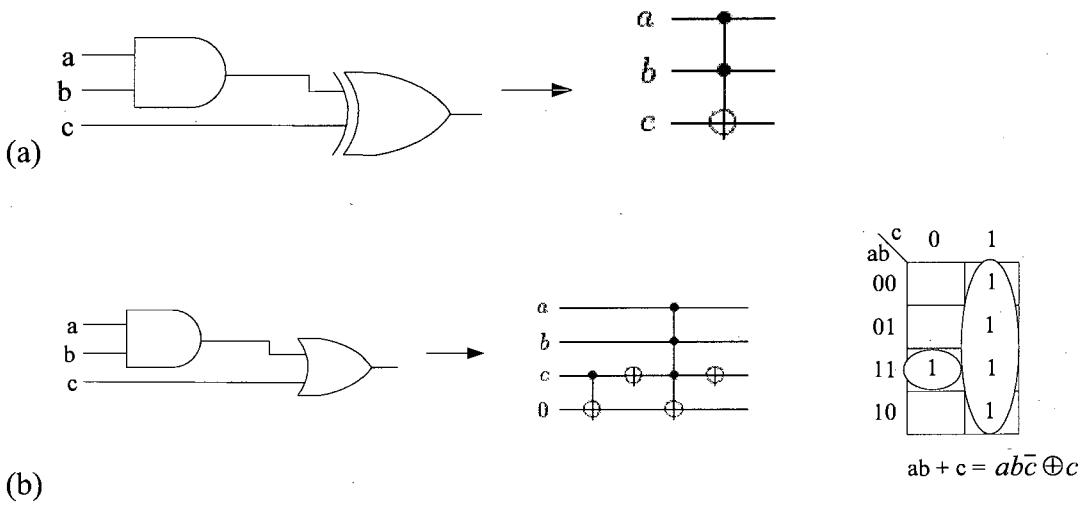
Figure 9.5.6.1: Realization of KRM form in a quantum array with separate functions f_1 and f_2. where $f = f_1 \oplus f_2$.

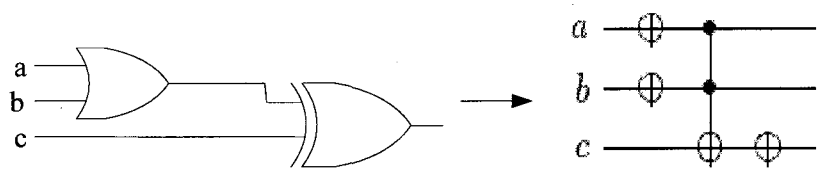
KRM-Like expansions can be realized in quantum arrays as in Figure 9.5.6.1. The first (from left) block includes negative polarity and mixed product literals and the second block the positive and mixed product literals.

9.5.7. Creating Base Functions Based on Bi-decomposition.

One more method to create quantum array is to use the classical bi-decomposition method. After this decomposition the circuit can be partitioned to node combinations and each node combination is realized by a quantum gate. Assume an arbitrary tree of 2-input gates that results from the bi-decomposition procedure [ref]. All possibilities of gate adjacencies of different types of gates are presented in Figure 9.5.7.1. Next all these small quantum gates are composed to larger arrays. Mirrors and copiers may be added. Some node combinations are given in Figure 9.5.7.1. An Example of

realization of quantum array using this method is presented in Figure 9.5.7.2, and the final circuit is given in Figure 9.5.7.3.

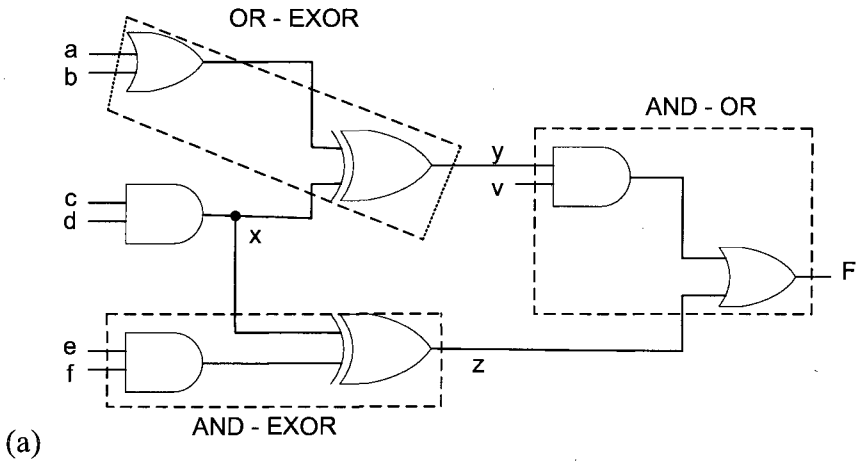




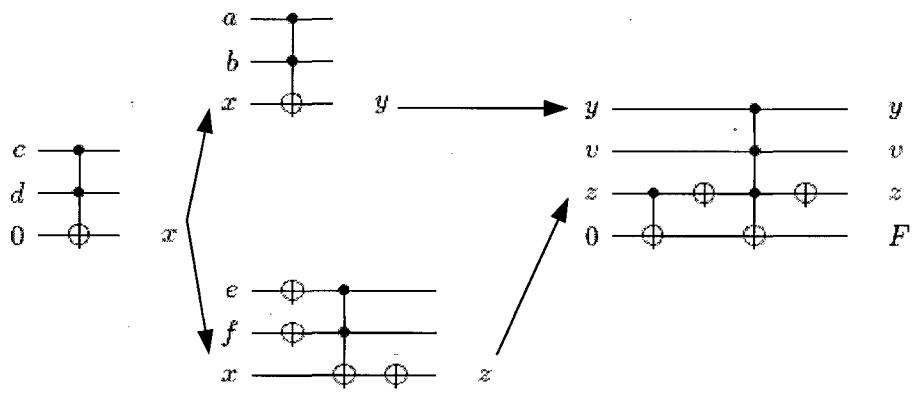
$$\overline{\bar{a} \bullet \bar{b} \oplus c} \quad = \quad \overline{\bar{a} \bullet \bar{b}} \oplus c \quad = \quad (a + b) \oplus c$$

(c)

Figure 9.5.7.1: Pieces of Quantum arrays corresponding to typical gate connections in classical bi-decomposition. (a) AND-EXOR node combination, (b) AND-OR node combination, (c) OR-Exor combination.



(a)



(b)

Figure 9.5.7.2: From Boolean bi-decomposition to quantum array. (a) Covering of a bi-decomposed circuit with patterns of node combinations, (b) each pattern is converted to its reversible equivalent.

Base on repeated decompositions of a function a set of base functions can be created and next enhanced to base families as in previous sections.

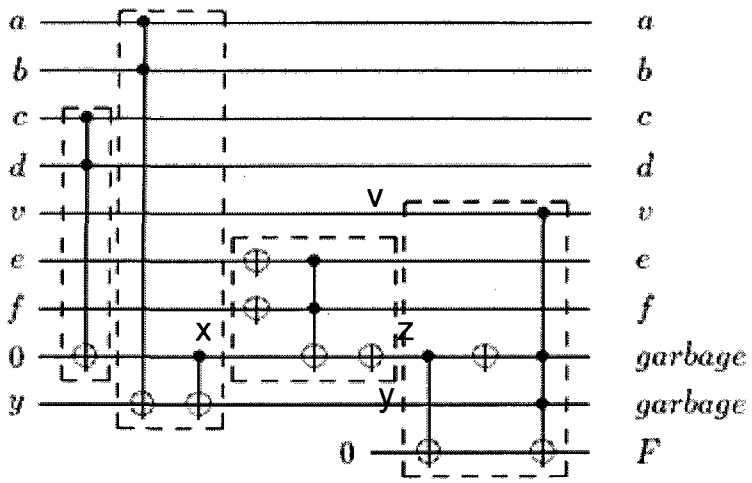


Figure 9.5.7.3: The final step of converting a bi-decomposed circuit to a quantum array. This circuit is created by laying out (the so-called quantum layout problem [Vijaya05]) of little reversible patterns into a large quantum array. This stage requires in general addition of SWAP gates and copying (Feynman) gates.

9.5.8. Composing Gates for Base Functions.

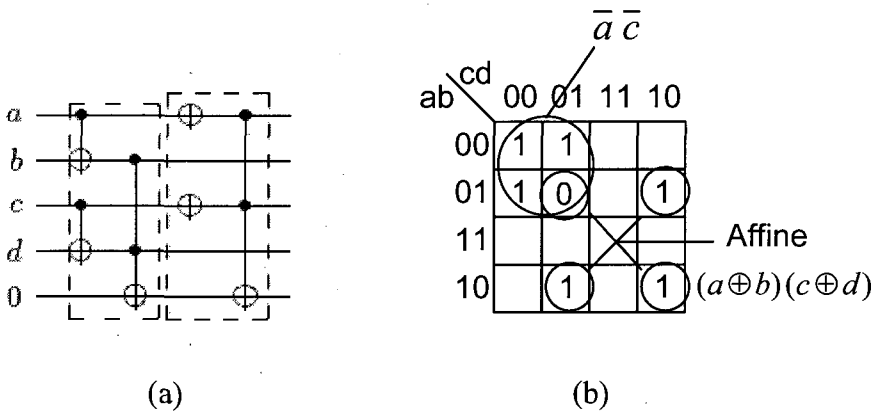


Figure 9.5.8.1: Realization of complex gates by composition. (a) Composition of Affine Toffoli gate and Toffoli gate, (b) the corresponding KMap. Note a "0" at the intersection of the $\overline{a}\,\overline{c}$ and the $(a \oplus b)(c \oplus d)$ patterns (groups of cells).

Complex gates can combine affine gates with standard Toffoli gate as illustrated in Figure 9.5.8.1. This way "more ones than zeros" heuristics and search methods from chapter 6 can be used to synthesize with LI bases.

9.5.9. Creating LI matrices for "all polarity search" algorithms from other LI matrices.

In Chapter 3, 4, 7 and 8 we discussed the algorithms of "all polarity search" types. A quantum algorithm for such search will be presented in Chapter 15. A question arises how to create base functions for this type of algorithms. We will present one possible answer to this problem below. This material relates also much to section 9.3.

Assume that matrix M1 transforms function F represented as a vector of minterms to its spectrum vector CV_1. Similarly, other matrix M2 transforms F to another spectrum vector CV_2.

$F \rightarrow M1 \rightarrow CV_1$
$F \rightarrow M2 \rightarrow CV_2$

Thus we have:

$$CV_1 = M1 \bullet F$$
$$CV_2 = M2 \bullet F \qquad \textit{(Equation 9.5.9.1)}$$

From where we get:

$$M1^{-1} \bullet CV_1 = F$$
$$CV_2 = M2 \bullet M1^{-1} \bullet CV_1 \qquad \textit{(Equation 9.5.9.2)}$$

$$CV_1 = M1_a \bullet M1_b \bullet F$$
$$CV_2 = N2_a \bullet M2_b \bullet F \qquad \textit{(Equation 9.5.9.3)}$$

$$M1_a^{-1} \bullet CV_1 = M1_b \bullet F$$
$$M2_b^{-1} \bullet CV_2 = M1_b \bullet F \qquad \textit{(Equation 9.5.9.4)}$$

$$M1_b^{-1} \bullet M1_a^{-1} \bullet CV_1 = F$$
$$CV_2 = M2_a \bullet M1_b \bullet (M1_b^{-1} \bullet M1_a^{-1} \bullet CV_1) \qquad \textit{(Equation 9.5.9.5)}$$
$$= M2_a \bullet M1_a^{-1} \bullet CV_1$$

Base on above equations one is able to create matrices such as $M2_a \bullet M1_a^{-1}$. Such matrices can be calculated once for all future uses. This approach allows next to find directly realizations for all polarities and this is done by just multiplying some matrices stored in the data base of matrices.

The algorithm based on this approach is the following.

Algorithm 9.5.9.1.

1. For functions of n variables given are all matrices M_i where each M_i corresponds to a family of base functions.
2. Find all matrices of type $M\_i2_a \bullet M\_i1_a^{-1}$ and store them in data base MM.
3. Given is vector F
4. By multiplying the first of matrices from MM by F calculate CV_1.

5. By multiplying the second of matrices from MM by CV_1 calculate CV_2.
6. Etc. iterate through all polarity matrices or their subset.
7. Find the solution with the smallest cost, i.e. the CV with most zero coefficients.

This method is easy to program in Matlab and very general. This method can be therefore applied to all families of base functions from chapters 3, 4, 7, 8 and 9.

The only drawback of Algorithm 9.5.9.1 is its relatively low speed.

CHAPTER 10

Affine Multiple-Valued Galois Gates and Their Circuit Structures

10.1. From Binary Affine Toffoli Gates to Affine Toffoli Galois Gates.

Binary affine functions are of a form LF $\oplus$ C where LF is a binary linear function and C is a binary constant (0 or 1). The same is true for ternary affine functions, but C = 0, 1, or 2. LF is a ternary linear function built from Ternary Feynman gates. The gate from Chapter 7, Figure 10.1.2, can be treated as a special case of an affine gate with the first column as the affine preprocessor and the last column as an affine postprocessor mirror. This kind of gate is a very powerful generalization of the binary Toffoli gate for any number of inputs. Can this design be further generalized to a new gate in Multiple Valued logic?

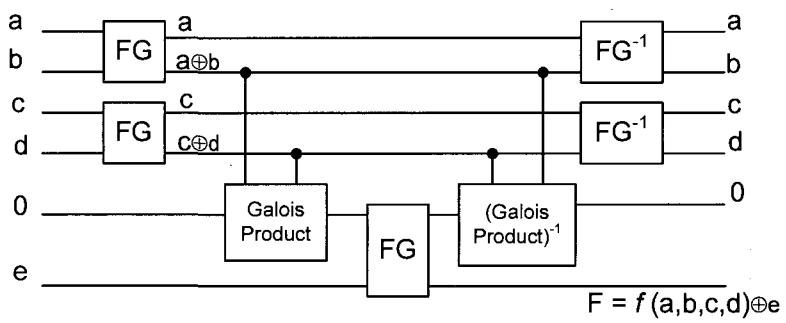


Figure 10.1.1: New (Affine Ternary) Toffoli gate which is a multiple-valued generalization of affine binary Toffoli gate for any radix K^m . FG stands for Feynman Galois gate (a ternary affine gate). This gate can be in particular realized in ternary quantum logic. Observe that one ancilla qubit initialized to $|0\rangle$ is necessary for Galois product gates for radix higher than 2 (the same number as in Boolean GF(2) case).

KMap of function f(a,b,c,d) realized by the gate F = f(a,b,c,d) ⊕ e, symbol ⊕ means Galois addition.

To answer this and similar questions, we will generalize now methods from Chapter 7 and other chapters to the ternary logic. We remember from the multiple-valued logic theory that standard AND/EXOR Boolean logic is the special case of the Galois Field Logic. In particular, in case of Galois Field (3) the addition operation is modulo 3 addition and the multiplication operation of this algebra is the modulo 3 multiplication. Therefore, our binary structure from Figure 10.1.1 has a direct counterpart in the Galois Field(3) circuit from Figure 10.1.2. Every Feynman Gate in binary (i.e. GF(2)) logic is replaced with the Feynman Gate (using modulo 3 addition) in ternary GF(3) logic. This is obvious since all axioms for GF(2) and GF(3) are the same. Similarly, every Boolean AND is replaced with the Galois 3 multiplication (the Galois Product). This leads directly to the circuit from Figure 10.1.2. This structure is very similar to its binary counterpart structure, thus allowing to re-use of all our ideas from previous chapters to synthesize ternary circuits. However, one should note that an ancilla bit was added to realize the Galois Product which is not a reversible operation. The question arises, is this ancilla bit absolutely necessary? If not, can we create a ternary Toffoli gate with no ancilla bits? Even if such gate would be created for 3 * 3 circuits, is this construction expandable with no ancilla bits for k * k ternary Toffoli Gates? This and similar questions will be the subject of considerations in Chapter 10. Another question is how to generalize these results from ternary logic to other radices, using either Galois Field circuit or some other type of multiple-valued

reversible circuits. Observe for instance, that the multiple-valued counterpart KMap of function *f*(a, b, c, d) (Figure 10.1.3) can be realized using the gate F = f(a, b, c, d) ⊕ e for any radix K , with K being a prime number, multiplication modulo K and addition modulo K.

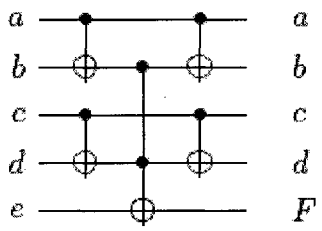


Figure 10.1.2: Binary Affine Toffoli Gate for function from Figure 10.1.3.

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

Figure 10.1.3: Graphical Analysis of the affine Toffoli gate from Figure 10.1.2. It uses product groups that are created by flattening of the formula originating for F directly from Figure 10.1.2.

10.2. Ternary Gates and Affine Ternary Gates.

10.2.1.Ternary Quantum Technology and Circuits

Classical computing has always been practically binary, although much research on ternary and general multiple-valued logic has been performed and experimental circuits have been fabricated. Quantum computing, among many other advantages, is a way to overcome the problem of the increasing percent of substrate (chip) area that must be devoted to connections only when the size of the design grows. This is because practical quantum computing can be multiple-valued and thus one wire (a qudit – ternary quantum bit) can transmit more information than a qubit.

The power of the binary affine gates is that they can be easily generalized to ternary affine gates and in general to multiple-valued quantum logic. This is done by just generalizing the binary Feynman gate to multiple-valued logic by replacing the EXOR operator with its multiple-valued counterpart. Because of the reversibility requirement this must be a group-based operation (in a sense of algebra), for instance the modulo-addition or the Galois Addition. While the Modulo Addition can be realized for any number (radix), the Galois Addition can be realized only for K^n where K is a prime number and $n \geq 2$ is a natural number. This may seem to be advantage of Modulo Addition as a base of MV logic, Galois Fields have however some other nice properties.

In [Giesecke07] we presented gates for ternary quantum logic. Synthesis of such gates and especially of larger circuits is still an open problem. We created exact minimum cost ternary reversible gates with quantum multiplexers using the method of iterative deepening depth-first search (IDDFS). Such exhaustive search approach is better for small problems than evolutionary algorithms or other heuristic search methods. Several new gates that have the provably exact minimum costs have been discovered. These gates are next used as library building blocks in the minimization of larger ternary quantum circuits like the highly testable GFSOP cascades [Khan03, Khan05] that generalize the binary ESOP cascades. These new gates can be also used as well in ternary circuits that generalize the so-called wave cascades [Mischenko02] from binary to ternary. These cascades were generalized by me in this chapter to ternary logic. The new gates are also useful to design oracles for multi-valued algorithms such as Deutsch-Jozsa [Fan07] and Grover [Fan07]. The optimally designed MV gates presented in this thesis can be next used as building blocks of larger gates in systematic synthesis methods which are extensions and generalizations of other previous logic synthesis methods that are now used in binary reversible and quantum circuits (Chapter 11 and [Miller06, Miller04, Al-Rabadi01, Khan05, Dubrova01, Al-Rabadi02, Lukac02, Khan03, Khan05 , Khan07, Denler04, Mishchenko02]). Alternately, one can use the exhaustive method to synthesize small circuits. Exhaustive search can be also used as a part of more sophisticated hybrid synthesizers [Lukac05]. The method searches exhaustively until the given circuit is found for

which it is next proven that within certain design constraints (like the size and the gate types) it is not possible to find a better realization of the given function F.

Ternary quantum macros – conceptual gates - can be implemented using quantum multiplexers [Perkowski02] as primitives. Quantum multiplexers are themselves composed from the Muthukrishnan-Stroud gates (M-S gates) invented by Muthukrishnan and Stroud and popularly used in designs [Muthukrishnan00]. The quantum multiplexer concept [Perkowski02] (called also the quantum mux) invented in [Perkowski02] and used also by several other authors, is a convenient intermediate notation to synthesize both binary and multiple-valued (mv) quantum circuits. Therefore, the synthesis in this chapter will be performed in terms of quantum multiplexers and their argument single-qudit functions. (Recall that qudits are quantum bits with radices higher than 2. Qutrit is a qudit used in ternary logic). Here I will introduce several different more or less regular structures that describe how these gates can be cascaded. We will find exact minimum solutions for some well-known logic operators and also for new gates in order to form libraries of universal gates for mv quantum circuit synthesis. The exhaustive search creates the gate as a cascade starting from input signals of the function and next adds sequentially quantum mux after quantum mux to create the logic outputs of the cascade. The first practical goal of the exhaustive search approach proposed here is to find the realizations of all 2-qudit-gates and determine their minimum costs and the best efficiencies. Efficiency can be defined in terms of how many ancilla qudits are used to realize a given functional

specification. Ternary quantum logic notation is based on the same principles as the binary Heisenberg and Dirac notations. The base vectors for ternary quantum logic are

$$|0\rangle = \begin{bmatrix}1\\0\\0\end{bmatrix}, \; |1\rangle = \begin{bmatrix}0\\1\\0\end{bmatrix} \text{ and } |2\rangle = \begin{bmatrix}0\\0\\1\end{bmatrix}.$$

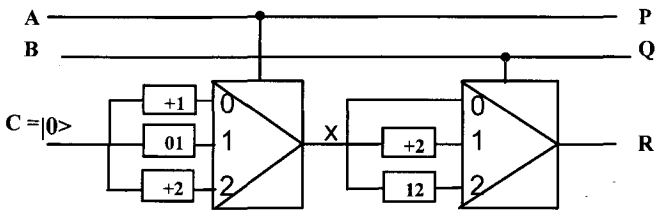


Figure 10.2.1.1: Example of implementation with ternary multiplexers. All additions are modulo 3. Operations +1, +2, (01) and (12) are single qutrit permutations. Please find the intermediate signal X.

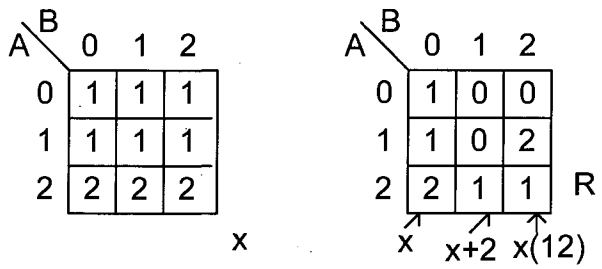


Figure 10.2.1.2: Graphical analysis based on ternary quantum multiplexers for the Example from Figure 10.2.1.1. The first map is for signal X. The second second map is for signal R. It has columns corresponding to single-qutrit operations: I (identity), +2 and (12), executed on single qutrit intermediate signal X from Figure 10.2.1.1.

Figure 10.2.1.1 shows a cascade implementation with two ternary quantum multiplexers. The small boxes at the left of the mux symbol (the symbol itself was taken from classical logic) represent arbitrary single qutrit unitary operators f_i, but in this thesis these operators are in addition permutative. The quantum multiplexer

operates as follows: depending on a value 0, 1, and 2 of the control qutrit, the respective input with number 0, 1 and 2, respectively (counted from top) is selected and sent to the output. Thus respective operator f_i is executed on the controlled qutrit. Figure 10.2.1.1 is a cascade of two multiplexers where A is the controlling qutrit of the first mux and B is the controlling qudit of the second mux. The target qudit C is initalized to $|0\rangle$. C is the controlled data qudit (qutrit in this particular case) on which the functions are applied. Operations +1 and +2 are implemented as cyclical shifts by 1 and by 2, respectively in one-qutrit operations. In Figure 10.2.1.1 let us look at the first multiplexer. Assume functions f_0, f_1, f_2, f_3, f_4 and f_5 to be defined as $f_0 = +1, f_1 = 01, f_2 = +2, f_3 = 02, f_4 = +2, f_5 = 12$. For circuit from Figure 10.2.1.1 the resulting ternary maps are shown in Figure 10.2.1.2. Operation (01) is a permutation of values 0 and 1 in a single qutrit, the operations (12) and (02) are implemented analogously. (These operations are realized internally by combinations of X, Y and Z Pauli quantum rotation operators [Giesecke07, Bae07] in data inputs of M-S gates [Muthukrishnan00]. They are inexpensive).

The motivation for the approach presented here is the realization of an arbitrary logic function through a series of cascaded gates. The synthesis goal is to minimize the number of ancilla qudits, while introducing greater freedom in the number of necessary stages. The number of ancilla qudits can only be hypothesized and depends always on the function to be realized. Muthukrishnan and Stroud [Muthukrishnan00] formulated a relation between the amount of data qudits and number of ancilla qudits

for M-S gates only. To reuse the ancilla qudits the circuits for our gates require in general mirror circuits (called mirrors for short). The implementation of mirror gates has the goal to restore the ancilla qudit to the initial state, $|0\rangle$, $|1\rangle$ or $|2\rangle$. The mirror gates G_i^{-1} use the inverse operations to their respective gates G_i. Thus $G_i^{-1} G_i = G_i G_i^{-1} = I$. Table 10.2.1.1 shows the corresponding inverse functions to the single-qutrit functions. Mirrors are used in MV wave cascades [Mischenko03]. They are also used in quantum realizations of GFSOP with various kinds of Toffoli-like and Feynman-like gates [Khan03, Khan05, Khan07]. They allow also to fold wires that start and end (thanks to mirror) with constants. Wire folding is based on graph coloring and similar algorithms known from classical logic/layout design. The mirror quantum multiplexers X and X^{-1} have inverse single-qudit functions f_i in all their data inputs.

Single qudit functions		Inverse single qudit functions
+ 1	→	+ 2
+ 2	→	+ 1
+ 01	→	+ 01
+ 02	→	+ 02
+ 12	→	+ 12

Table 10.2.1.1: Inverse functions for each single qutrit function.

10.2.2. Ternary Galois Field Logic, Reversible Gates.

As it is known, the Galois Toffoli gate together with the single qubit rotations (permutations) create a universal ternary logic system [Kerntopf04, Brylinski01]. It is also convenient, using an analogy to binary reversible logic, to add the Ternary Feynman gate to this system.

Any unitary matrix represents a quantum gate (this property is true for any radix of logic). As we remember, if a unitary matrix has only one *"1"* in every column and the remaining elements are *"0"*'s, then such a matrix is called a permutation matrix. The set of output vectors of such a permutation gate is simply a certain permutation of the set of input vectors. A practical realization of MV Feynman gate was shown in [Muthukhrishnan00]. Our exhaustive algorithm found first the minimal realization based on the quantum multiplexer structure for 2-qudits (+ 1 ancilla qudit) gates. The cost function for mux cascades was defined by the total number of single-qubit operations used. For the ternary Feynman realization only two multiplexers and 4 single-qudit-functions are needed as well as one ancilla qudit. Since the Feynman gate is a permutation gate, based on results from [Yang06] the algorithm was used to search for a solution without any ancilla qudits. Figure 10.2.2.1 illustrates such a solution. Instead of using two quantum multiplexers and 4 single-qudit operations, only one mux and 2 single-qudit operations (*+1 and +2*) are required. The mux number is one and the cost is 2 operations. Interestingly, the number of muxes of the ternary Feynman gate, which is one, is the same as for the binary Feynman gate.

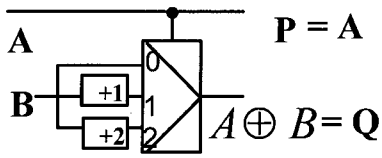


Figure 10.2.2.1: Realization of the Ternary Feynman gate using one quantum multiplexer and two single qudit operations. An important fact is that no ancilla bits are required which makes it a great gate for affine extensions.

In addition we used adaptations of exhaustive methods (similar to those from chapter 7) to find useful ternary circuit solutions. These methods gave no assurances of circuit minimality, but they took into account certain important practical constraints such as the user-specified limited number of ancilla qudits.

Ternary Feynman is used in all ternary gates and circuit structures (forms) in place of binary Feynman gates in binary gates and circuits. This way linear ternary circuits are created and with constants 1 and 2 affine ternary circuits are created. Let us observe that Ternary Feynman gate is our name – Richard Feynman did not invent this gate. We gave this name by analogy based on Galois Field similarity of binary and ternary circuits. This analogy is an obvious result from the structural/algebraic similarity of GF(2) and GF(3) fields. The ternary Feynman gate can be viewed as a Galois Field 3 Addition in which both inputs A and B are added up. The Ternary Galois Field (we call it also TGF, or GF(3)) consists of the set of elements $T = \{0, 1, 2\}$ and two basic operations – **addition** (denoted by +) and **multiplication** (denoted by · or absence of any operator). Muthukrishnan and Stroud [Muthukrishnan00] found a relation between

the number of qudits and the number of required ancilla qudits. This formula is based on the MS gates, which are different from the universal quantum multiplexer used in the present thesis, $r = [(n-2)/(d-2)] \quad (d > 2)$. The number of ancilla qudits is r, whereas n is the number of data qudits and d is the radix of the logic. For ternary logic d is 3. For example a system with 7 qudits needs 5 ancilla qudits to perform its logic function.

Efficient methods for representing and minimizing Ternary Galois Field Sum of Products (TGFSOP) expressions are very important. Such expression can be either realized directly in quantum cascades or it is a starting point of factorization processes leading to factorized cascades and wave cascades [Mishchenko02, Khan03]. These methods are not a subject of this dissertation and can be studied for example in [Khan05, Al-Rabadi01]. It should be stressed however, that the results of this dissertation contribute the cheapest gates to be used in both the Galois Field Sum of Product (GFSOP) architecture and the factorized GFSOP cascades. We invented also their ternary affine generalizations.

The other GF(3) operation is the Galois (Field) Multiplication (Figure 10.2.2.2) (called also Galois Product – Figure 10.1.1). While the Galois addition replaces EXOR, the Galois multiplication can replace the Boolean AND operation in the multivalued domain [Curtis04] and is therefore a very important operator in quantum circuit design. But it is only one method of AND operator generalization to ternary logic and not a unique ternary gate that must be used for this task.

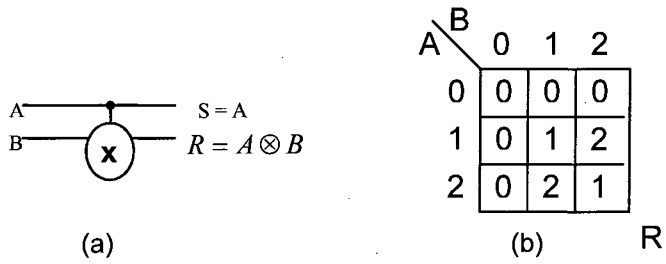


Figure 10.2.2.2: Galois Field (3) multiplication; a) a symbol. b) the ternary map which shows that GF multiplication is not a Latin Square.

If input $A = 2$ and $B = 2$ then the output R yields to $R = 2 \otimes 2 = 1$ as the result of modulo 3 product. The realization of the Galois multiplication in our approach is given in Figure 10.2. 2.3.

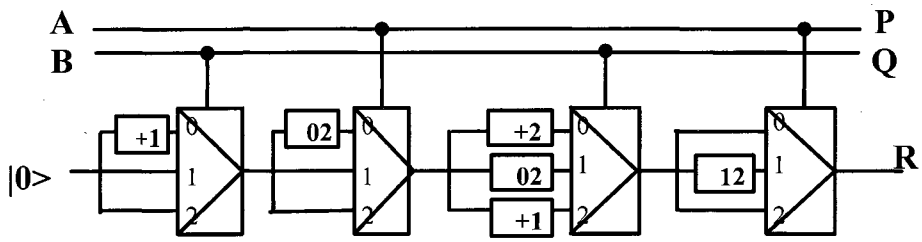


Figure 10.2.2.3: Realization of Galois field multiplication using quantum multiplexers. Note that one ancilla qutrit is used. It is not possible to have no ancilla qudits as the GF multiplication is not a group operator.

For the realization of the Galois Field multiplication operation 4 multiplexers with 6 single-qudit operations are needed. One ancilla qutrit is needed.

An interesting Toffoli-like ternary gate in Galois Logic [Khan03, Khan05, Perkowski02] that uses both the Galois multiplication and Galois addition was found

by our software and presented in Figure 10.2.2.4. We call it the Ternary Galois Toffoli gate.

(a)

AB \ C	0	1	2	A • B
00	0	1	2	0
01	0	1	2	0
02	0	1	2	0
10	0	1	2	0
11	1	2	0	1
12	2	0	1	2
20	0	1	2	0
21	2	0	1	2
22	1	2	0	1

(b)

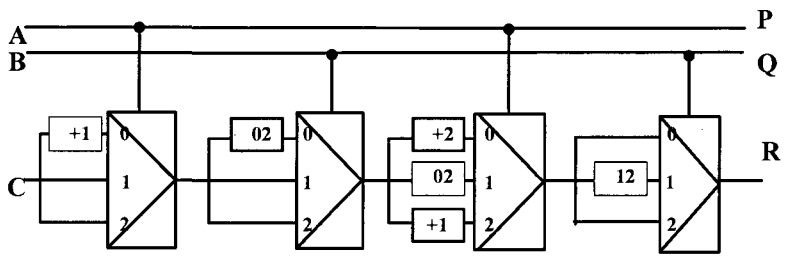


Figure 10.2.2.4: Ternary Galois Toffoli (2-controlled-NOT) gate; minimal solution using quantum multiplexers. No ancilla qudits are used. Note only 4 single qutrit operators. This gate is a natural and elegant generalization of binary Toffoli to the ternary quantum logic found by software.(a) the Ternary Map and the product AB, (b) the circuit.

This way, our software proved exhaustively that a universal system for Galois Field(3) logic can be created with 5 single-qutrit permutation gates, Galois Feynman and Galois Toffoli gates and Galois multiplication as an additional gate. Every ternary function (reversible or not) can be realized in our system, using ancilla bits for ternary non-reversible functions. The minimum universal systems [Brylinski01, Kerntopf04] for ternary quantum permutative logic are still an open but not very practical problem.

This is because non-minimal sets of gates can be easily created which improve the quality of results.

10.2.3. Ternary SWAP Gates.

The importance of SWAP gates is known in binary quantum logic. Based on examples of ternary quantum oracles that I considered I found that these gates play the same useful role in the ternary quantum logic. Such gate when added to any universal system allows to reduce the size of the circuits, sometimes dramatically. Therefore we asked our software to design the SWAP gate. The result is shown in Figure 10.2.3.1a. The graphical analysis is shown in Figure 10.2.3.1b. The analysis confirms that the automatically designed circuit is correct. The 2-qudit SWAP gate exchanges two qudits. It has no counterpart in the classic electrical domain, because the swapping is simply done by crossing wires within two layers of metalization. In the domain of quantum computing a "crossing" of wires is not possible. If the prototype of a SWAP gate for 2 qudits is developed, it can be used for circuit with n-qudits. The swapping is then just performed between 2 qudits within this circuit. A general approach to swap a given number of qudits might be an interesting research subject. There are, for example, 6 possible input/output combinations for a 3-qudit SWAP. Only 2 combinations are real 3-qudit SWAP gates and all swaps can be build with only two SWAP gates. Binary counter part of such gates are used excessively in quantum oracles (see Chapters 11, 12, 13, 14 and 15).

More realizations of the SWAP gate can be found in [Giesecke06]. Khan *et al.* found a solution for the 2-qudit SWAP gate in [Khan07], our solution made with the exhaustive search algorithm is the same and we proved that it could not be improved (assuming the design with muxes). Interestingly, both in binary and ternary cases the SWAP gate has three muxes.

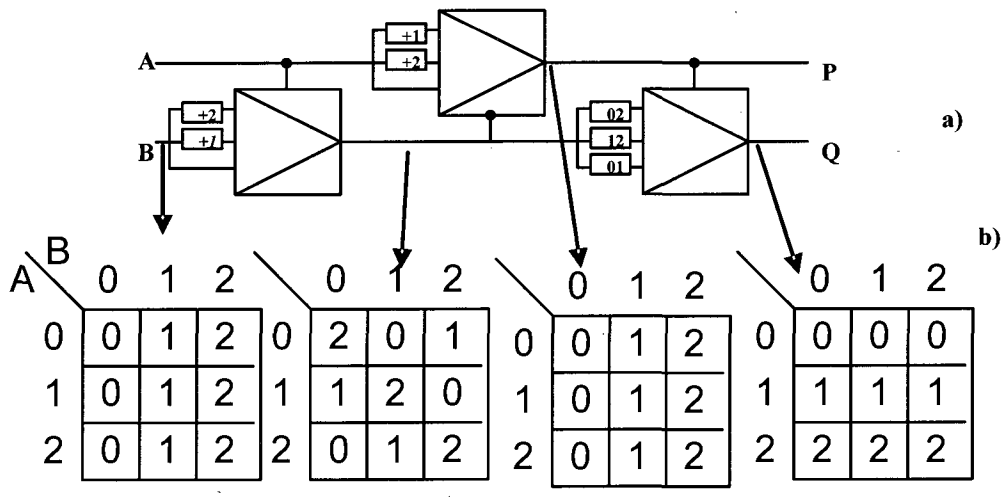


Figure 10.2.3.1: (a) The structure of the Ternary SWAP gate and (b) the graphical analysis using ternary logic maps. The ternary maps (generalizations of KMaps) are drawn to help the Reader to analyze this circuit.

Based on this gate, I found some general properties which I used next to built SWAP gates also for other radices, especially for radix 4, but I do not discuss this in the thesis.

10.2.4. Realization of classical MIN/MAX multiple-valued logic and their generalizations circuits in ternary quantum circuits.

In classical binary logic AND gate and OR gate are the well-known gates. As a standard, in multiple-valued logic domain the AND gate is replaced by the MIN gate and the OR gate is replaced by the MAX gate. The MIN gate is the arithmetic minimum and the MAX gate is the arithmetic maximum of integers being their arguments. When the respective maps were entered into our exhaustive search algorithm the results were that both these quantum gates need 6 single-qudit operations. The algorithm found these solutions with the order "DCDC" of the control variables. An interesting fact to notice is that the order can also be switched to "CDCD" order and no changes are made on the single-qudit operations and the gate outputs the same result. This results from the symmetry of maps (order of arguments can be changed) and from the reversibility (see Figs 10.2.4.1, 10.2.4.2 and 10.2.4.3).

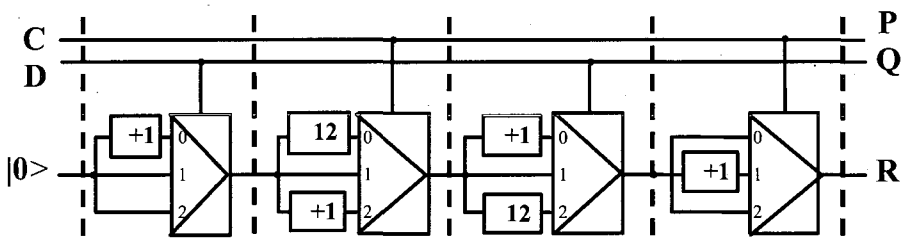


Figure 10.2.4.1: Realization of the Ternary Min gate with control order of "DCDC".

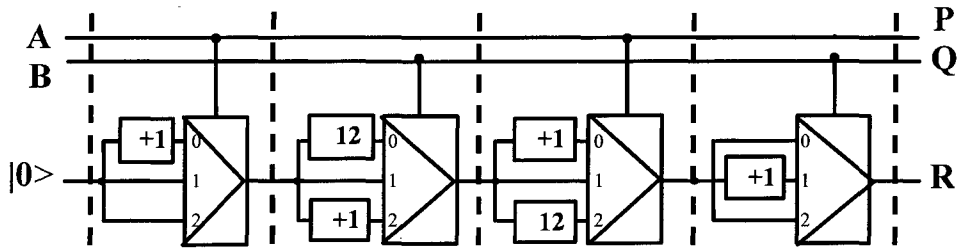


Figure 10.2.4.2.: Realization of the Ternary Min gate with control order of "ABAB".

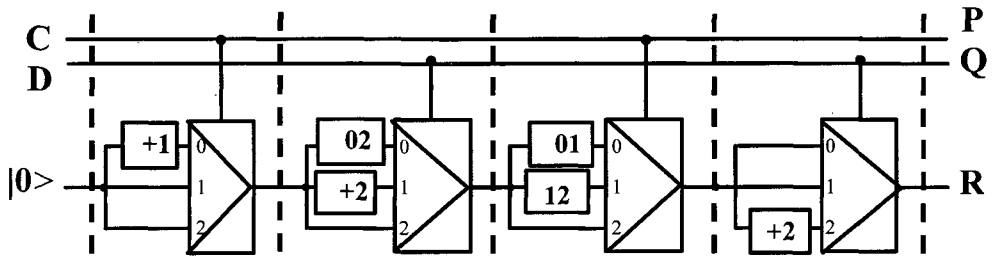


Figure 10.2.4.3: Realization of the Ternary Max gate with control order of "CDCD".

We have found an efficient way to implement all two-qudit ternary quantum gates by having at most one ancilla qudit. The ancilla qudit is the only drawback of this type of synthesis. For all ternary functions (operators) that are originally not reversible the ancilla qudit is needed in any case, and thus should not be viewed as a weakness of our approach. Simply nothing better can be found than found by the exhaustive search software.

The MIN gate together with gates that are the controlled single-qudit permutations produce powerful circuits for any radix. Figure 10.2.4.4 realizes a simple cascade in which the MIN/MAX ternary logic is combined with controlled ternary gates which

control single qutrit permutations in the lowest qutrit with symbol d. The target gates in the lowest bit d are controlled with some selected value of the control bit, for instance 2. If the qutrit is in this state, then the target operation is executed. Otherwise the gate is an identity. In this way, because of using ancilla bits, any ternary control function can be created. The target gate operation can be any single-qutrit (reversible) operation that it available. This is for instance one of five single-qutrit permutations in case of the ternary logic. (There are six permutations but identity is useless).

The gate design process as presented in this section is very general and applies to any radix of logic, since the MIN, MAX, and MODSUM gates and the controlled permutations can be realized for any radix of logic. Moreover, ternary affine polarity and standard (ternary) polarity pre- and post-processors can be added in MV logic in exactly same way as they were added in binary cases and discussed previously in Chapters 7, 8 and 9.

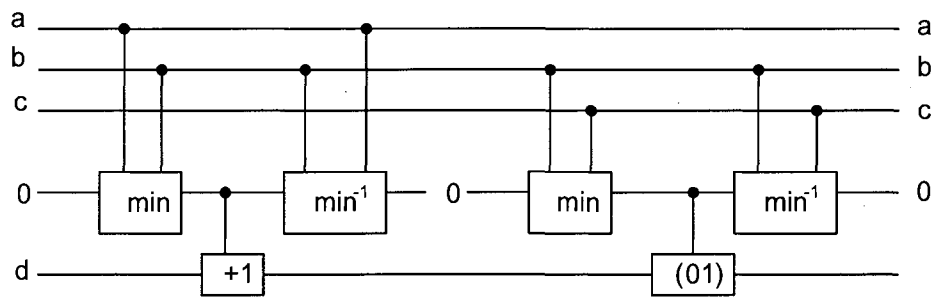


Figure 10.2.4.4: A cascade of two 2-controlled Toffoli-like gates for ternary logic that uses the ternary minimum operator. The controlled target can be any single-qutrit permutation. The use of the inverse (mirror) circuits returns the ancilla qubit to zero and thus decreases the width of the circuit. Therefore only one ancilla qutrit is used.

Less than one ancilla qudits is not possible because the MIN gate is not reversible as MIN operator is not a group operator .

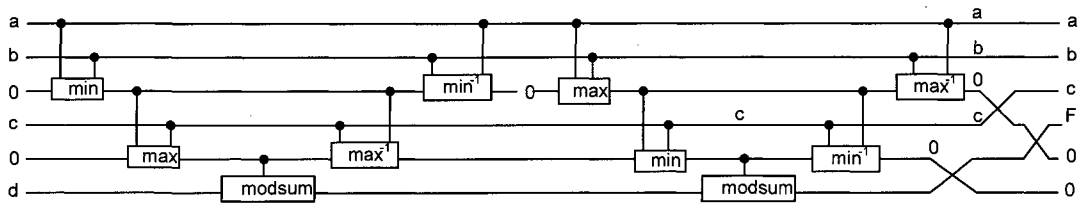


Figure 10.2.4.5: Ternary Wave Cascade. This circuit uses Ternary-Controlled Ternary-Target Gates, where MODSUM operator is used as the addition operator (using all ternary values). Ternary SWAP gates are also added to have a required order of outputs in the oracle.

Figure 10.2.4.5 illustrates a way of combining MIN/MAX and MODSUM logic using also mirror gates and SWAP gates. It is a Modsum-based cascade of ternary Maitra cascades. This circuit generalizes the well-known binary Maitra cascades and Wave Cascades. In addition the use of SWAP gates is illustrated. This circuit, not known from literature, generalizes to ternary quantum logic the well-known concept of the binary Maitra cascades and also the reversible wave cascades of Mishchenko and Perkowski [Mishchenko02].

Figure 10.2.4.6 presents a classical MIN/MAX logic realization (one stage only) using ancilla qudits in target MAX gates. This is a ternary quantum generalization of the standard classical binary SOP-like logic for which much is published and for which available software exists.

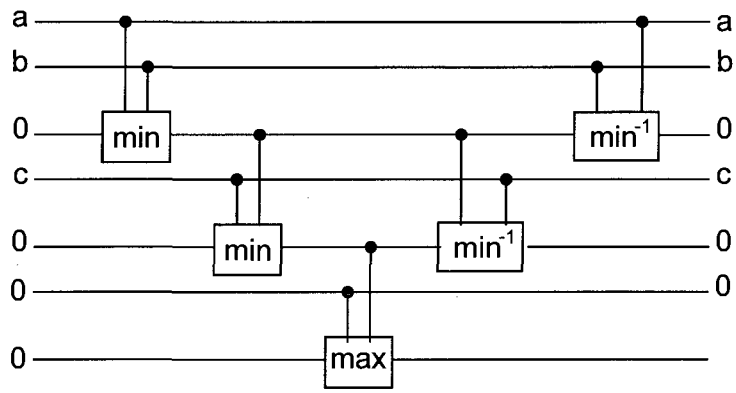


Figure 10.2.4.6: Classical MIN/MAX logic realization (one stage only) using ancilla bits in target MAX gates.

Figure 10.2.4.7 presents the affine generalizations of these cascades (one segment and affine processor only). Section 10.3 will discuss the generalization of these types of circuits to hybrid quantum circuits in which qudits may have different radices.

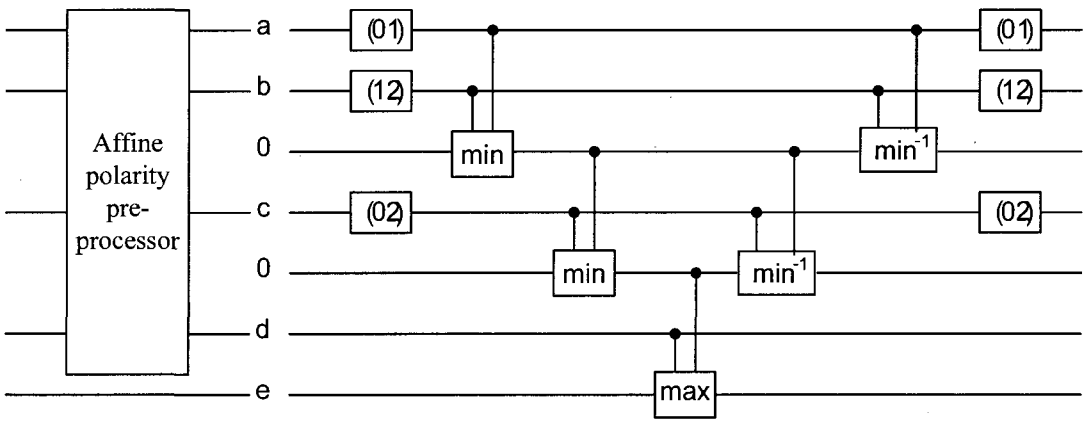


Figure 10.2.4.7: Affine generalizations of reversible cascades for MIN/MAX logic (one segment and affine pre- processor shown only).

Although in this dissertation we will not work on the synthesis algorithms for these new structures, one should stress that all methods and algorithms from this thesis can be generalized to them in a straightforward way. The universality of these circuit structures results directly from fundamental results and properties of ternary logic. Gates MIN, MAX, Galois Product, Galois Sum and Toffoli can be freely mixed in ternary cascades.

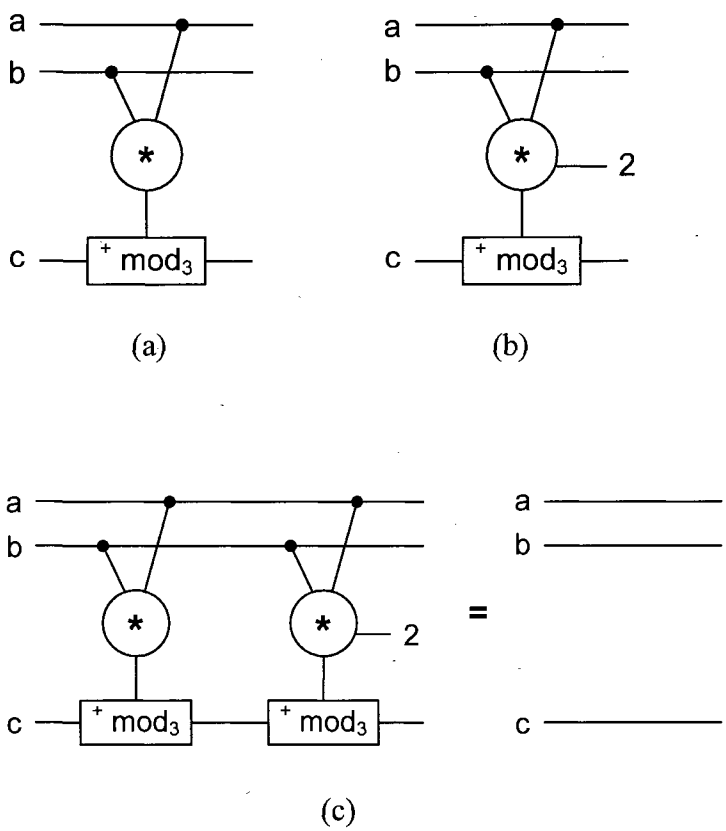


Figure 10.2.4.8: Creation of mirror circuits in ternary logic. (a) Galois Toffoli gate and its (b) "Inverse Galois Toffoli" gate that multiplies additionally by constant 2, (c) verification that Inverse Galois Toffoli gate is an inverse of Galois Toffoli gate. ($ab\ ^{+}mod\ ab \bullet 2\ ^{+}mod\ c = ab \bullet 3\ ^{+}mod\ c = c$).

a \ b	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

a • b

a \ b	0	1	2
0	0	0	0
1	0	2	1
2	0	1	2

a • b • 2

Figure 10.2.4.9: Ternary maps of the a • b and a • b • 2 operators.

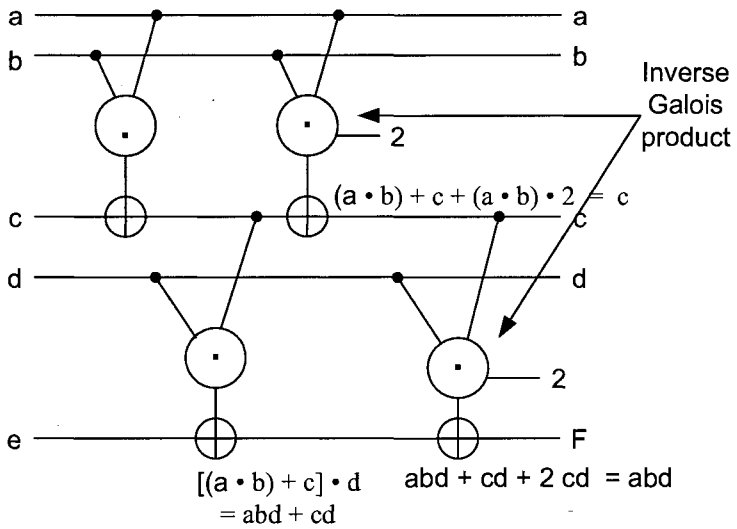


Figure 10.2.4.10: Using of mirrors in ternary Galois cascades that realize big ternary Galois Toffoli gates.

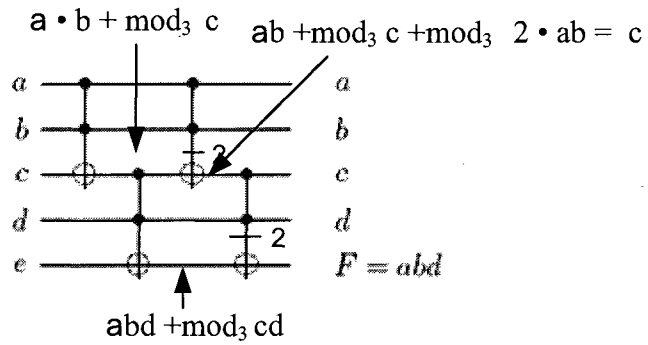


Figure 10.2.4.11: Simplified schematics with ternary notation for the circuit from Figure 10.2.4.10.

The design of mirror circuits for ternary logic is slightly more complicated than in binary logic where $F^{-1} = F$ for every NOT, Feynman and Toffoli gate. In ternary logic the mirror must be such F^{-1} that $F \bullet F^{-1} = I$. Thus the inverse of gate from Figure 10.2.4.8a is shown in Figure 10.2.4.8b. The matrices of a • b and 2 • a • b are given in Figure 10.2.4.9. Thus $ab \ ^{+}mod_3 \ 2\ ab = 3\ ab = 0$, hence the identity from Figure 10.2.4.8c. Thus the circuit from Figure 10.2.4.10 can be rewritten to a simplified notation from Figure 10.2.4.11.

Concluding, Galois Logic circuits are very similar to binary circuits discussed in this thesis. This result is not accidental since from the beginning it was my intention to work on affine gates, because they have cheap group group-theoretical extensions in multi-valued logic. The beauty of the developed by me methods is they can be all extended to Galois Field Circuits and many of these methods can be extended to MIN/MAX and hybrid circuits.

10.2.5. Synthesis of Polynomial Circuits Based on Galois Field Gates.

In this section I present my new algorithm for ternary cascades based on Galois Field Logic. I use GF Toffoli, GF Feynman and five single-qutrit permutations.

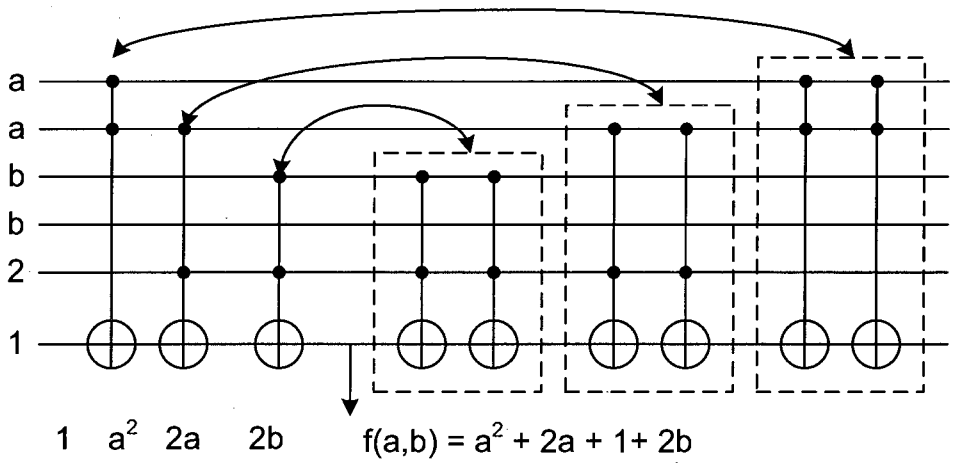


Figure 10.2.5.1: Example of Realization of a ternary polynomial in a quantum cascade with mirrors.

The algorithm of Agrawal and Jha can be extended to MV logic and in particular to ternary logic, which I have done. Let us first look at Figure 10.2.5.1 where the realizations of several simple ternary polynomials are given. Observe repeating qubits for the same variable and using mirrors which are duplicated original gates.

a	0	1	2
^0a	1	0	0
^1a	0	1	0
^2a	0	0	1
$a.a = a^2$	0	1	2
$a+1 = a^2+1$	1	2	0
$a+2 = a^2+2$	2	0	1
$(a^2+1).(a^2+2)$	1	0	0
$a^2(a^2+1)$	0	1	0
$a^2(a^2+2)$	0	0	1
2a = a + a	0	2	1
2a + a = 3a	0	0	0
2a + 1	1	0	0
2a + 2	2	1	0

Figure 10.2.5.2: Some Ternary polynomials of single variable.

Figure 10.2.5.2 illustrates some basic functions - ternary polynomials of a single variable. They are represented as ternary polynomial expressions (column one) and as ternary maps of universal Post Literals (column two, three and four).

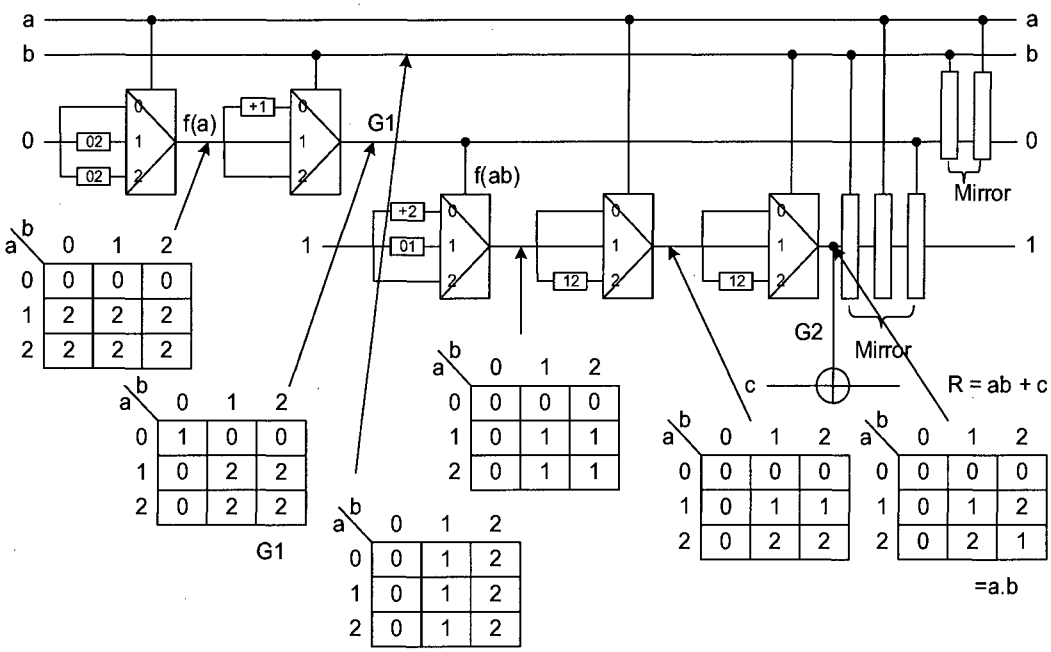


Figure 10.2.5.3: Analysis/Synthesis of Galois Field(3) Toffoli using single-controlled ternary quantum multiplexers with 2 ancilla qubits.

My first challenge was first to design the GF(3) Toffoli gate. I first built it from ternary quantum multiplexers as in Figure 10.2.5.3. This first hand design was worse than the result from Figure 10.2.2.4 which was found subsequently by our exhaustive software. In some insight I found next by hand a better than the first one solution which is shown and explained in Figure 10.2.5.4. It is difficult to calculate the exact costs of circuit from Figure 10.2.2.4 and Figure 10.2.5.4 without calculating pulses, but we have no easy method to do this.

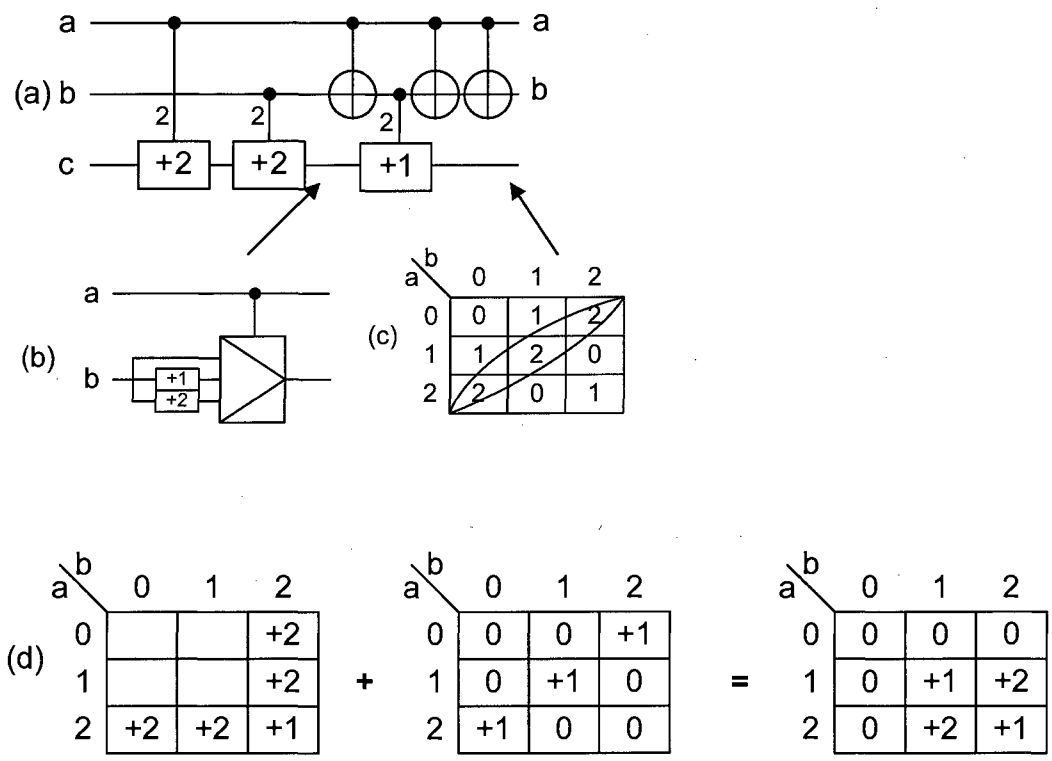


Figure 10.2.5.4: (a) Realization of Ternary GF Toffoli from M-S gates and Ternary Feynman gates, (b)Ternary Feynman from ternary mux, (c) Ternary KMap of ternary Feynman gate, (d) realization of Galois product as a composition of "+2"controlled gates (left map) and controlled "+" gate (middle).

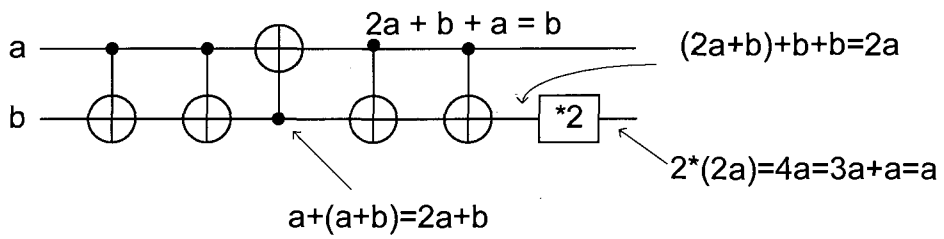


Figure 10.2.5.5: Realization of ternary SWAP using ternary Feynman gates, single qubit operators and ternary GF(3) polynomials. The solution is analyzed and synthesized using ternary GF polynomials.

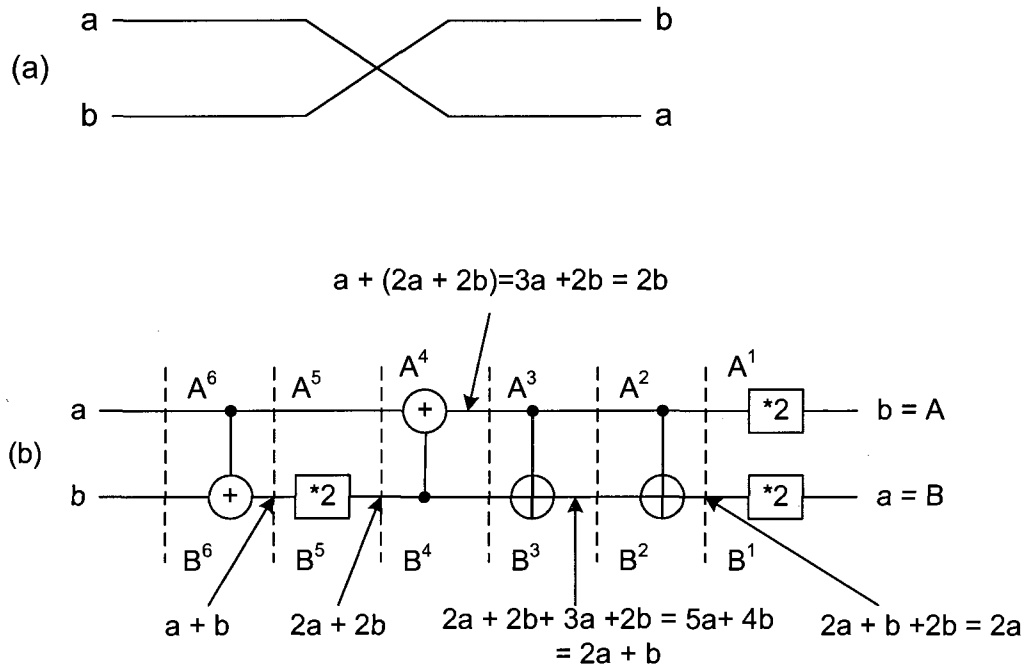


Figure 10.2.5.6: (a) Symbol of ternary Swap gate, (b) its realization with annotated expressions showing stages of analysis or synthesis based on ternary GF polynomials.

Playing with gates and their ternary maps, I found next the realization of SWAP gate from Figure *10.2.5.*5 using ternary Feynman gates. I found that *2 gate is needed. Next using M-S gates I found a better solution for ternary SWAP, illustrated in Figure *10.2.5.*6b.

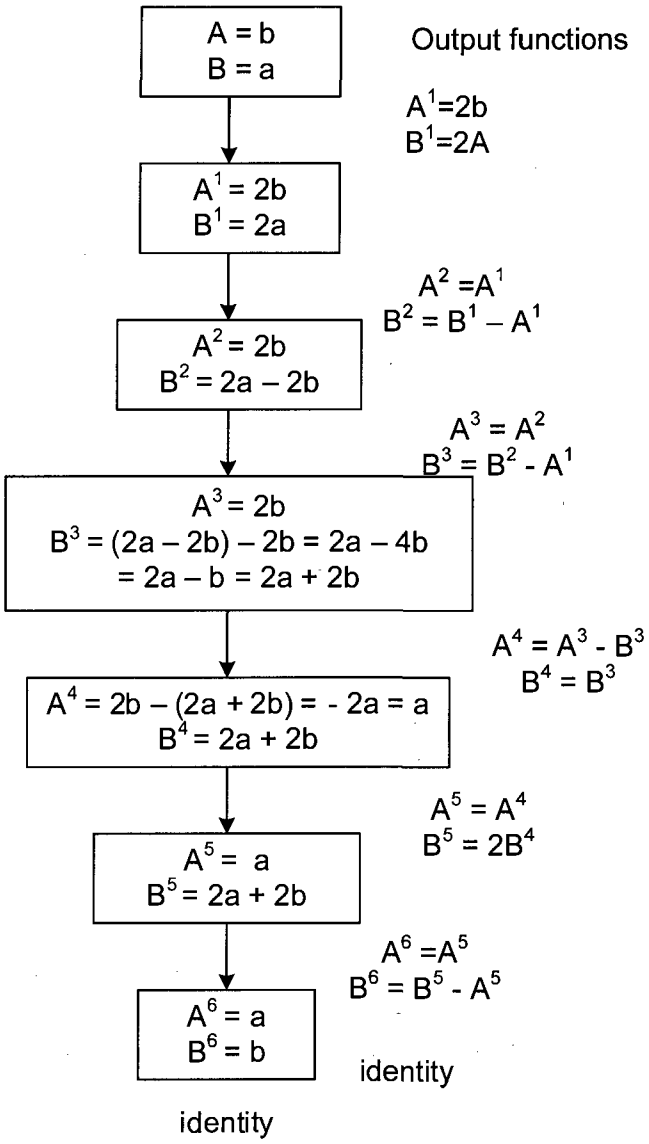


Figure 10.2.5.7: Synthesis of Ternary SWAP gate from outputs to inputs using ternary polynomials.

This last invention and its schematics annotated with GF(3) polynomial expressions lead me to the generalization of the Agrawal/Jha's algorithm for the case of ternary logic, as illustrated in Figure 10.2.5.7.

Figure 10.2.5.7 presents the process of synthesizing the ternary SWAP gate from outputs to inputs. The idea of the Agarwal/Jha algorithm has been here extended to ternary logic and also generalized to all ternary single-qutrit gates and SWAPs.

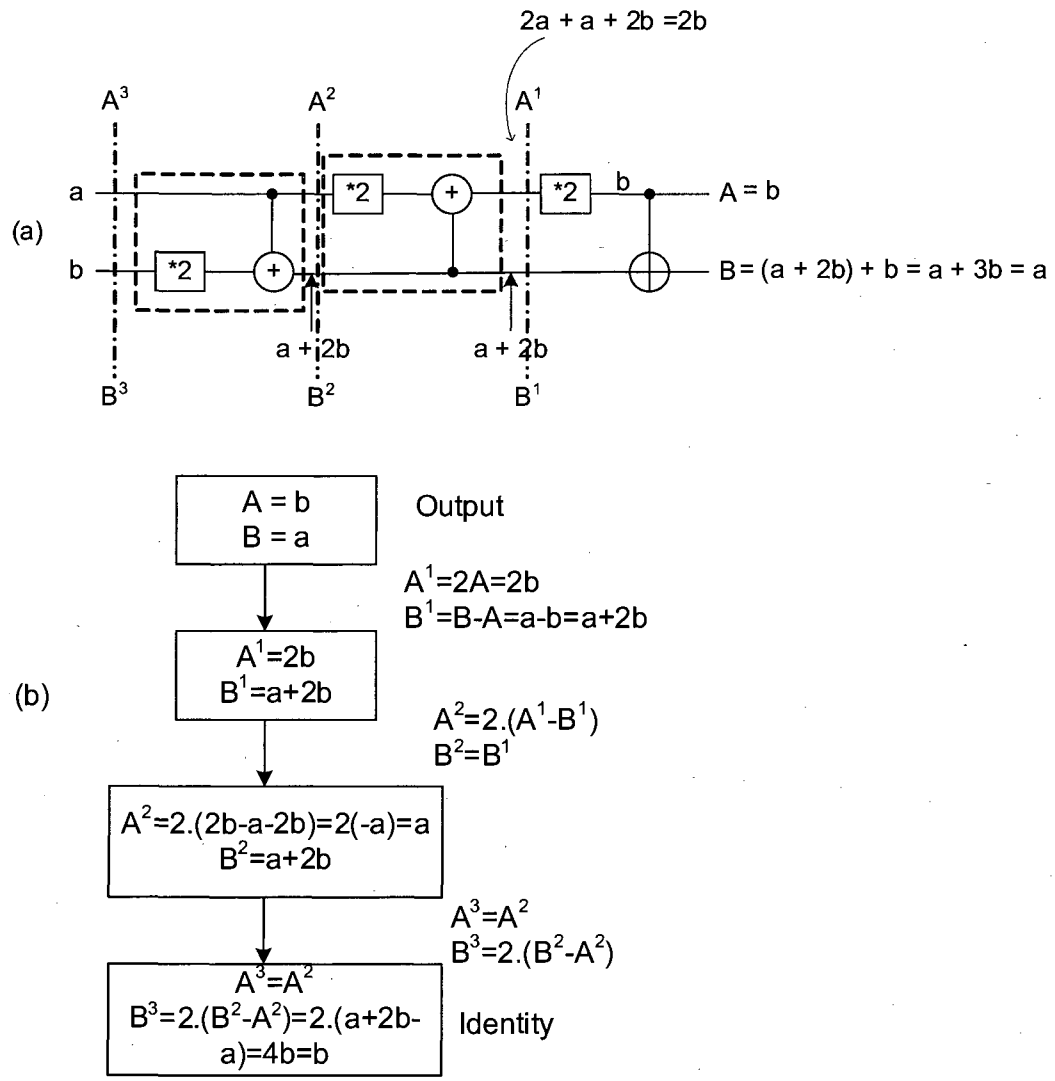
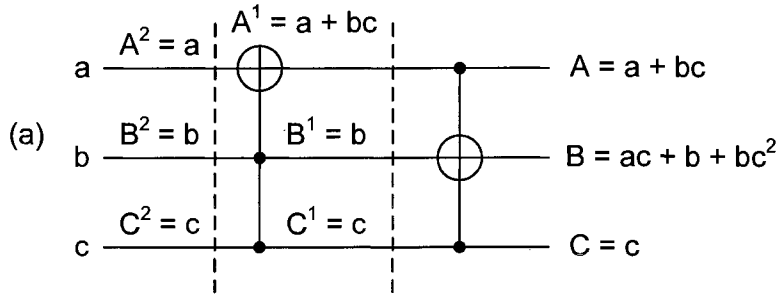


Figure 10.2.5.8: Using polynomials to synthesize ternary SWAP gate (second search process for the same gate). (a) the annotated ternary GF circuit, (b) the branch of the search tree.

Using my new theory extensions of Agrawal/Jha I found a better solution to Ternary SWAP, illustrated in Figure 10.2.5.8.



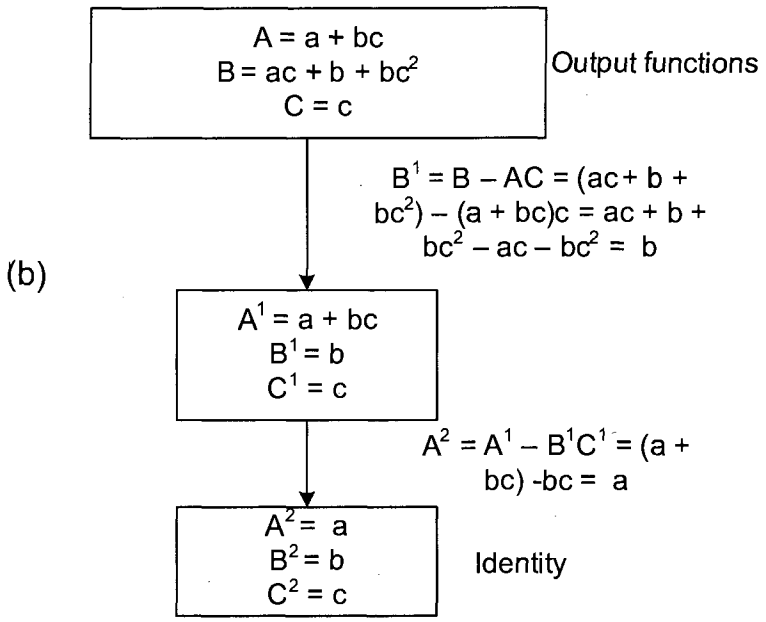


Figure 10.2.5.9: Using factorization method of Galois Field expressions for synthesis of a GF(3) reversible circuit.

Figure 10.2.5.9 b presents the synthesis process of another ternary circuit from outputs to inputs using this extended method. The resulting circuit using Ternary GF Toffoli gates obtained from this process is shown in Figure 10.2.5.9 a.

10.2.6. Realization of new type of Toffoli gates in ternary quantum logic.

Here we will design a Toffoli-like ternary gate that is not a Toffoli Galois Field gate. While in binary permutative logic there exists only one 2×2 gate which is CNOT, there are many its counterpart 2×2 gates in the ternary logic. So which ones are good? The exhaustive algorithm produced for instance the realization of the new ternary Controlled-NOT gate shown in Figure 10.2.6.1. Here the controlling value has been selected to 2, but similar gates with arbitrary control values can be created. Observe that this gate is also a generalization of the binary Feynman gate, but it is different from the group-based generalization of the Feynman gate used in both the Modulo-Addition logic and the Galois-Addition logic. Obviously, this gate is easy to generalize to arbitrary radix and is very similar to the well-known Muthurkrishnan/Stroud gate. There exist very many gates similar to this one, with other 1-qudit operations on inputs to the quantum mux.

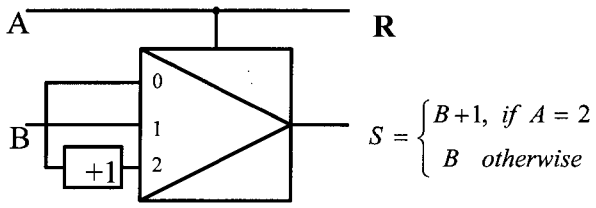
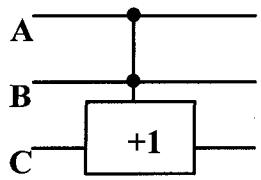


Figure 10.2.6.1: Realization of the Ternary Controlled-NOT gate. Value 2 of qudit A-R is selected here as the activating value.

Can one derive from the structure of the Toffoli gate or Toffoli-like gate a quantum circuit that might not require an ancilla qudit? With our exhaustive algorithm an implementation of the Toffoli as the Controlled-Controlled-NOT gate with 3 data qudits and without any ancilla qutrits is possible. Binary Toffoli is known as universal [Yang06, Miller06] and is therefore another important gate. The same should be in ternary. Yang *et al.* [Yang06] show that a system of SWAP, NOT and Controlled-Not is universal for the realization of arbitrary ternary *n*-qudit reversible circuits without an ancilla qudit. From the Toffoli gate, which is a 2-Controlled-NOT, it is possible to build up an n-qudit Controlled-NOT. The Toffoli gate is a *controlled-controlled-(+1)* gate and its diagrammatic representation is presented in Figure 10.2.6.2.



$$Q = \begin{cases} (C + 1) \bmod 3, & \text{if } A = 2 \wedge B = 2 \\ C & \text{otherwise} \end{cases}$$

Figure 10.2.6.2: Symbol of Ternary Toffoli of "if-then-else" type gate and its function. Observe that this gate is different from the ternary Galois Toffoli gate from section 10.2.2. This is also denoted by ${}^{2}A^{2}B\,(+1C)$ where ${}^{2}A$ is the post literal.

This gate activates "+1" operations in target qudit *C,* if and only if the qudits *A* and *B* are both *"2"* and leaves *C* alone if they aren't (i.e, it is an identity in such case). The controlling qudits *"A"* and *"B"* remain unchanged and are simply mapped to the output. The exhaustive search algorithm found 7128 ways to realize various variants of ternary Toffoli gates. One of the minimal solutions is shown in Figure 10.2.6.3. The cost of this realization is 4; meaning only four single qudit operations are needed. Interestingly, in terms of the number of quantum multiplexers this gate needs only 4 muxes, while the binary Toffoli gate needs 2 Feynman gates, 2 Controlled-V and one Controlled-$V^{\dagger}$ for a total of 5 muxes (Controlled-V is also called Controlled-Square-Root-of-NOT, Controlled-$V^{\dagger}$ is its Hermitian [Bae07, Nielsen00]). It should be obvious that two affine generalizations (as previously) can be done as well for this type of gates. Moreover, any of 5 1-qutrit permutation gates can be controlled by A and B in Figure 10.2.6.2. There are therefore five new generalizations of Toffoli (non-Galois) to ternary logic. Analysis of circuit from Figure 10.2.6.3 is in Figure 10.2.6.4.

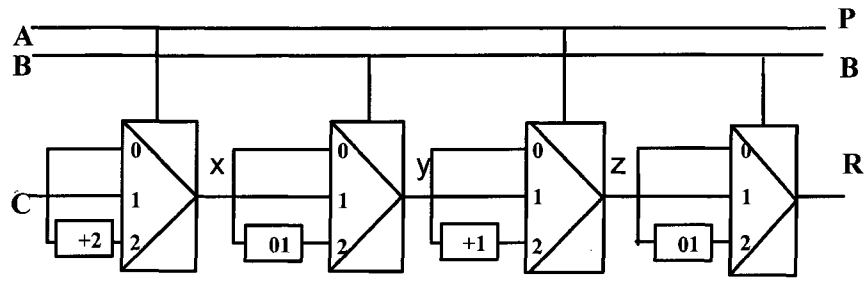


Figure 10.2.6.3: Ternary Toffoli (2-controlled-NOT) gate for the symbol from Figure 10.2.6.2. This is the minimal solution using quantum multiplexers.

AB \ C	0	1	2
00	0	1	2
01	0	1	2
02	0	1	2
10	0	1	2
11	0	1	2
12	0	1	2
20	2	0	1
21	2	0	1
22	2	0	1

X

AB \ C	0	1	2
00	0	1	2
01	0	1	2
02	1	0	2
10	0	1	2
11	0	1	2
12	1	0	2
20	2	0	1
21	2	0	1
22	2	1	0

Y

AB \ C	0	1	2
00	0	1	2
01	0	1	2
02	1	0	2
10	0	1	2
11	0	1	2
12	1	0	2
20	0	1	2
21	0	1	2
22	0	1	2

Z

AB \ C	0	1	2
00	0	1	2
01	0	1	2
02	0	1	2
10	0	1	2
11	0	1	2
12	0	1	2
20	0	1	2
21	0	1	2
22	2	1	0

R

Figure 10.2.6.4: Analysis of the first new Toffoli gate which has a "+1" operator in target bit.

10.3. Affine Hybrid Gates with Binary Outputs.

Hybrid gates are similar to the above ternary gates but can have a different logic radix in every quantum wire (qudit). The special case of such gates which are of interest in oracle design are gates with multiple-valued inputs and binary outputs. The outputs of these gates can be used directly in certain Boolean logic functions (usually global AND realized in multi-input Toffoli) in oracles. Figure 10.3.1 presents some hybrid gates with control qudits that are ternary and target qubits that are binary. Synthesis with such gates should be similar to the synthesis with ternary gates from section 10.2 since because the hybrid gates are special cases of the ternary gates. Also, replacing in the lowest bit of the cascades the general MODSUM gate with the EXOR gate will produce hybrid circuits of this type (Figure 10.3.4). Both types of affine generalizations can be also applied to these circuits. Examples of these types of circuits are given in Figures 10.3.2, 10.3.3 and 10.3.4.

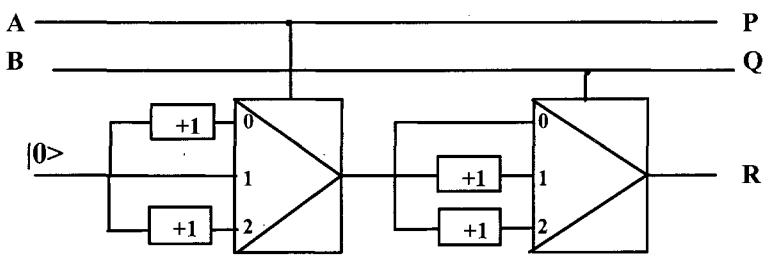


Figure 10.3.1: Realization of ternary-control binary-target hybrid quantum circuit using quantum multiplexers. Control qubits A and B are ternary and qubit with output R, the target, is binary. As we see, the circuit is very similar to ternary circuits, the only difference is that the target single-qudit operators on the target qudit (qubit in this case) are only binary operators, wire and +1(mod2) operator which is equivalent to binary NOT.

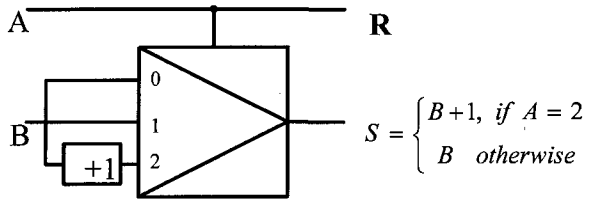


Figure 10.3.2: Realization of the Ternary Controlled-NOT gate with binary target. Observe that this is exactly the same diagram as in Figure 10.3.1 above but here the qudit B is binary. Because qubit B is binary B + 1 (mod 2) = NOT (B).

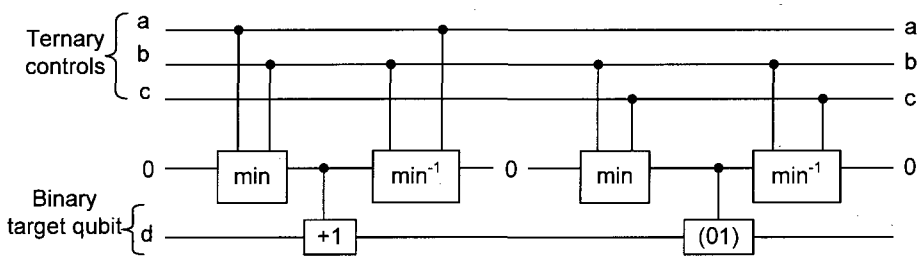


Figure 10.3.3: A cascade of two 2-controlled Toffoli-like gates for Modulo sum of minima type of circuits. Observe that this is exactly the same diagram as in Figure 10.3.4 earlier in text but the interpretation of operators in the target qubit with input d is different. Now the signals in qubit d are binary, so both operation +1 and (01) are interpreted as standard binary inverters, activated by any non-zero control value.

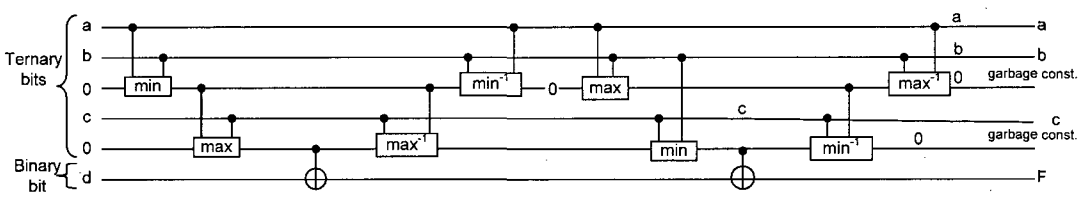


Figure 10.3.4: Ternary-Controlled Binary-Target Hybrid Wave Cascade structure.

Observe that some gates, such as SWAP, are not realizable in hybrid technology. Hybrid SWAP is just not possible to exist, because if the lower qudit has only two values how can it swap values with the upper qudit that is ternary?

10.4. Extending Zhegalkin Hierarchy.

The expansions, trees, decision diagrams, and forms in the Zhegalkin Hierarchy are tabulated in Table 10.4.1. Diagrams and forms can be created from trees in the standard way [Perkowski97a]. For each of these structures a quantum array can be created, as we illustrated by many examples in chapters 3, 7, 8 and 9. In chapters 7, 8 and 9 we introduced also affine gates that lead to new "Affine Forms". I add therefore the column of Affine Forms (the last column, at right) to the Hierarchy Table as my original contribution to the Zhegalkin Hierarchy.

Expansion	Tree	Diagram	Form	Affine Forms
Single Polarity RM Expansion	Zhegalkin Single Polarity Tree (ZRMT)	Zhegalkin Decision Diagram	Single Polarity Zhegalkin Forms	Single Polarity Zhegalkin Affine Forms
Any subset of Zhegalkin Expansions, but only one type in every level	Zhegalkin Kronecker RM Tree (ZKRMT)	Zhegalkin Kronecker RM Decision Diagram	Zhegalkin Kronecker Reed-Muller Forms (ZKRM)	Zhegalkin Kronecker Reed-Muller Affine Forms
Any subset of Zhegalkin expansions in a level, for ordered variables	Zhegalkin Pseudo Kronecker RM Tree (ZPKRMT)	Zhegalkin Pseudo Kronecker Decision Diagram (ZPKDD)	Zhegalkin Pseudo Kronecker Reed-Muller Forms (ZPKRM)	Zhegalkin Pseudo-Kronecker Reed-Muller Affine Forms
Any subset of Zhegalkin expansions with any order of variables in each branch	Zhegalkin Free Kronecker RM Tree (ZFKRMT)	Zhegalkin Free Kronecker Decision Diagram (ZFKDD)	Zhegalkin Free Kronecker Reed- Muller Form (ZFKRM)	Zhegalkin Tree Kronecker Reed-Muller Affine Forms

Table 10.4.1: Extended Zhegalkin Hierarchy table with new "Affine Forms".

Finally, let us observe that all concepts introduced in this thesis are based on algebraic axioms. Axioms and algebras related to this dissertation are given in Table 10.4.2 and Table 10.4.3. These formulas demonstrate the generality of all shown here Linearly Independent Logic methods for various MV logics. They are useful to derive expressions and design synthesis algorithms. Quaternary circuits were not included in the thesis.

	a+0=a	a.**1**=a	ab=ba	Flattening ab+ac=a(b+c)	a+**1**=**1**	a+a'=**1**	a+a=0	a'=**1**-x a.a'=0	De Morgan
Boolean algebra	+	+	+	+	+	+	+	+	+
GF(2) = AND/EXOR	+	+	+	+	no	+	+	+	no
GF(3)	+	+	+	+	no	+	no	no	no
Min modsum 3	+	+	+	+	no	+	no	no	no
GF(4)	+	+	+	+	no	+	+	no	no
Ring 4	+	+	+	+	no	+	+	+	no
Min/max	+	+	+	+	+	+	no	+ a'=**1**-x	+

Table 10.4.2: Comparison of basic algebra axioms used in various algebras related to this dissertation. Columns correspond to axioms and rows to algebras. Symbol + means that given axiom belongs to given algebra. Bold **1** *is the unity symbol of logic.*

ALGEBRAS	Expansion	Unity **1**	a+-a=0 axiom	$a.a^{-1}=1$ for $a \neq 0$ axiom
Boolean algebra	Shannon (this thesis)	**1**=1	no for +	yes
GF(2) = AND/EXOR	Davio Shannon (this thesis)	**1**=1	yes for +	yes
GF(3)	Davio Shannon (this thesis, [Mozammel])	**1**=1	yes for mod 3 addition -a = (0–a)	yes for GF (3) multiplication $a^{-1} = a$
Min modsum 3	(this thesis, Dipal Shah)	**1**=2	yes for mod 3 addition	no for min
GF(4)	[Mozammel]	**1**=1	yes for Galois addition	yes for GF (4) multiplication
Ring 4	[Dipal Shah]	**1**=3	yes for modulo 4 addition	no for ring multiplication
Min/max	[Dipal Shah]	**1**=radix -1	no for max	no for min

Table 10.4.3: This table illustrates relations between algebras and expansions used in LI logic. The third column compares the 1 element in various logics. The fourth and fifth columns compare the use of group axioms for the addition and multiplication operations, respectively, in all these algebras. This table teaches us that because of axioms, most of the introduced in chapter 10 synthesis concepts are applicable only to Galois Fields.

10.5. Conclusions.

In this chapter we introduced the concept of multiple-valued affine functions and operators, and we showed examples of applications of this concept to design ternary and hybrid gates called affine MV Toffoli gates. These concepts can be expanded to

quaternary and general multiple-valued logic, and also to the hybrid logic of mixed radices proposed for the first time in this dissertation.

In standard Grover Algorithm the cascade is binary. It will be shown in Chapter 11 however, on certain arithmetic blocks for oracles that the cascades can have also multiple-valued inputs and binary outputs. Finally, even binary input – binary output oracles can have some restricted multiple-valued sub-blocks inside them (like the "Count Ones" block from Chapter 11). Therefore, it was important to create general concepts of affine gates that extended to ternary the binary affine concepts from sections 7.1- 7.4 in Chapter 7.

Various ternary and hybrid permutative gates and synthesis methods can be used to create the quantum cascades as introduced in this chapter. The choice of the gate types and their realization using quantum-realizable primitives are thus problems of basic importance to both binary and multiple-valued quantum logic synthesis algorithms. In this chapter we introduced affine gates, first ternary and next hybrid – ternary/binary as a starting point to create gates, polarized expansions and general-purpose circuit structures.

We showed in this chapter that exhaustive search makes it possible to generate all possible ternary two-variable output functions, using, at worst, one ancilla qudit and only four quantum multiplexers. Exact minimum solutions have been found in

particular for the ternary MIN, MAX, Feynman, Galois addition and Galois multiplication, Toffoli, and SWAP gates. Many more such generalizations are possible [Giesecke06, Giesecke07]. We found two ternary generalizations of Feynman and two generalizations of Toffoli, Galois, new Toffoli (not Galois), both useful as building blocks for various types of quantum cascades. Using this method, the ternary (new) Toffoli gate (not Toffoli Galois) has been realized with 4 quantum muxes equipped with four single-qudit operations. The program proved also that all 2-qudits gates can be realized within at most 4 quantum muxes and only one ancilla qudit. The method allows to investigate trade-offs between the number of gates and ancilla bits. For instance, a circuit without ancilla bits may be theoretically realizable but would likely be much longer than a circuit with one ancilla qudit. The related question of synthesis is a difficult one and open to future research. For instance, in [Yang06] it was proved that ternary SWAP, NOT and 1-Controlled-NOT gates are universal for realization of arbitrary ternary n-qudit reversible circuits without ancilla qudits. Can these results be extended to our new gates? Paper [Yang06] also demonstrated that all even ternary n-qudit reversible circuits can be constructed by ternary NOT and ternary 1-Controlled-NOT. Moreover the method from [Yang06] is constructive, which means that it can be programmed to obtain the circuit for any number of qudits. However, the circuits according to [Yang06] seem to be unnecessarily long. Our new method from this chapter allows to compare circuits with the minimum number of ancilla bits with those that have few more ancilla bits. Other approaches, not referenced here for a lack of space were based on various evolutionary and Nature-mimicking paradigms [Yang06,

Bae07]. Although these methods found several large circuits as well as circuits presented here, they were not able to find any new realization of a universal quantum gate of the smallest cost. For instance, none of these methods did deliver results for 3 qudit gates like the 3-qudit SWAP and ternary Fredkin gates yet. The iterative deepening depth-first search method is more practical for these tasks than the biology-mimicking methods. The full potential of Iterative Deepening has been not yet fully recognized in quantum circuits community. It can be combined with A* search algorithm by adding a heuristic evaluation function as discussed in chapter 6. An interesting open problem is to extend these search methods to arbitrary radix logic.

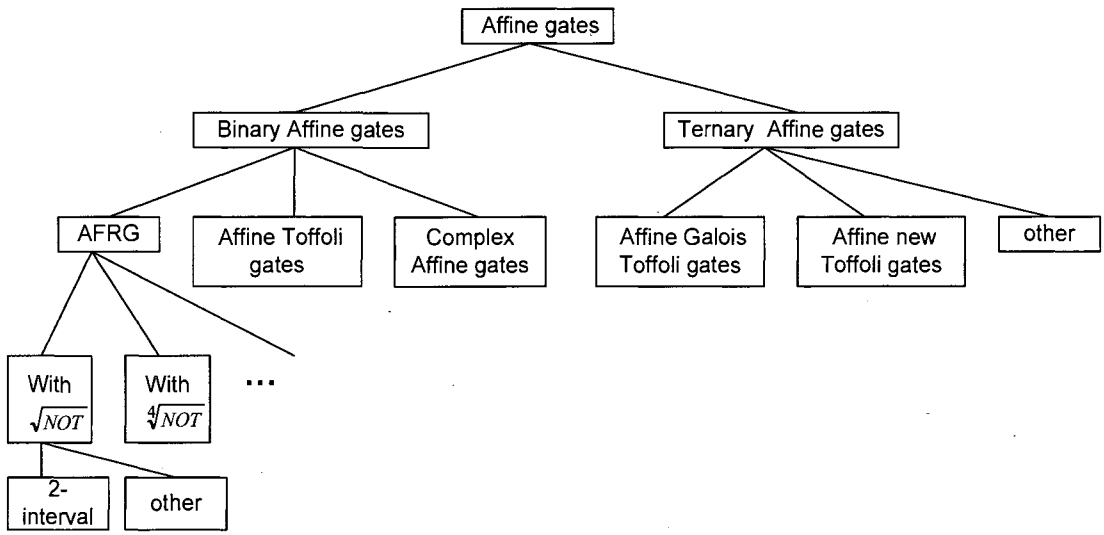


Figure 10.5.1: Partial Classifications of affine gates (for Binary and Ternary logic).

Finally all the structures discussed in chapters 3, 4, 7, 8, 9 and 10 can be enhanced by adding affine gates and their pre-, post-processors. Figure 10.5.1 gives the

classifications of affine gates, both binary and ternary. Similar classification can be done for hybrid gates.

Concluding this chapter, we showed very powerful generalizations to Reed-Muller logic that encompass very large spaces of circuit structures. This requires even more sophisticated search algorithms to find high quality solutions.

Observe that there are several unifying and generalizing design themes in this thesis:

1) from search to quantum parallel search,
2) from standard oracles to quantum oracles,
3) from classical logic circuits to AND/EXOR circuits to reversible circuits to quantum circuits,
4) from affine binary circuits to affine multiple-values circuits,
5) from standard polarity to affine polarity,
6) from LI families to affine LI families.

The themes related to gate and circuit design culminated in this chapter. This completes the circuit-related part of my dissertation.

CHAPTER 11

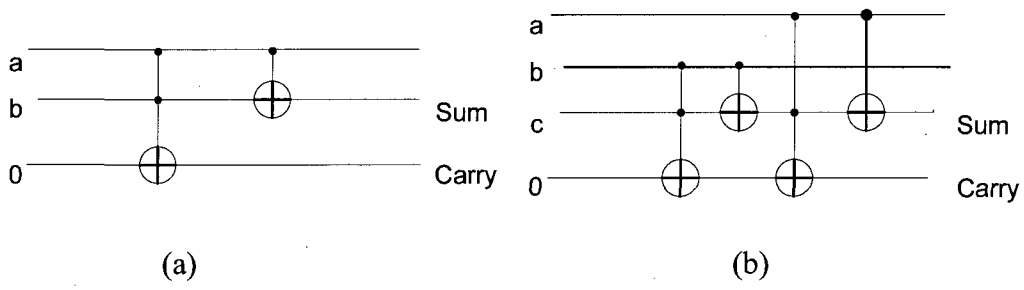
Design of Blocks for Oracles and Quantum Computers using Permutative Circuits

11.1. Introduction

In this chapter we show how the synthesis methods introduced in chapters 3, 4, 6, 7, 8, 9 and 10 can be used to design useful blocks for reversible and quantum oracles. We design binary and multiple-valued-input circuits. In some cases, we use synthesis methods developed by us in previous chapters. We will introduce also less formalized methods that are however useful in hand synthesis of quantum oracles. This chapter is a link between the methods of circuit design (chapters 2, 3, 7, 8, 9, 10) with the methods of algorithms (oracles) design (chapters 12 - 15). Below, the quantum blocks are presented in the order that emphasizes their mutual connections, rather than in groups of circuits of the same type or application.

11.2. Simple Adder Circuits.

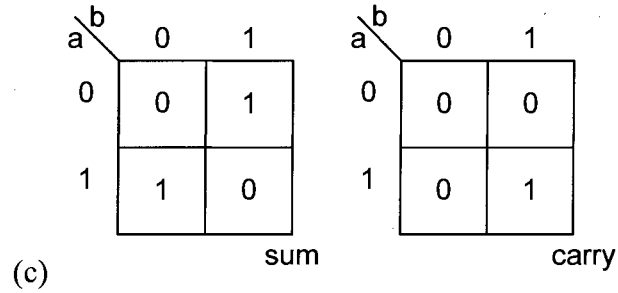
The Quantum implementation for half-adder is shown in Figure 11.2.1a. Circuit for full-adder is shown in Figure 11.2.1b. The circuit from Figure 11.1a can be obtained using any method from chapters 3, 4, 6 – 9 and the circuit from Figure 11.2.1b can be obtained by the method from chapter 3 that creates sequentially outputs of a multi-output function converting at the same time the irreversible function to a reversible function in the process.



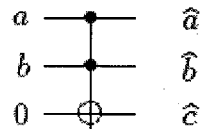
(a) (b)

Figure 11.2.1: Quantum Adders. (a) Half-adder HA realized using Toffoli and Feynman gates, (b) Full adder FA realized using two Toffol gates, one Feynman and one inverter gates. Sum = a ⊕ b ⊕ c, Carry = a (b ⊕ c) ⊕ bc.

The synthesis process of the half-adder is illustrated in Figure 11.2.1.c – g. Figure 11.2.1c has KMaps of sum and carry functions. The Carry output is realized first from left in the array because it requires an ancilla bit anyway as an irreversible function and it does not affect the state of inputs. Thus, after the realization of carry, the circuit looks as in Figure 11.2.1d. Now, Figure 11.2.1e presents a truth table for the new synthesis problem with inputs $\hat{a}, \hat{b}, \hat{c}$ and output sum. When converted to a KMap (Figure 11.2.1f) we can observe don't care's. This is one of the reasons that we consider don't cares in this thesis, and especially in chapters 7 – 9. The function from Figure 11.2.1f is next realized as a linear function Sum = $\hat{a} \oplus \hat{b}$ in Figure 11.2.1g which completes the synthesis process of the circuit from Figure 11.2.1a. After realizing first the carry output as $\hat{c}$ (Figure 11.2.1d) the outputs of the gate become new variables $\hat{a}$ = a, $\hat{b}$ = b, $\hat{c} = ab \oplus 0 = ab$ for the next stages of synthesis. Thus the circuit from Figure 11.2.1a has been realized.



(c)



(d)

(e)

ab	$\hat{a}$	$\hat{b}$	$\hat{c}$	Sum
00	0	0	0	0
01	0	1	0	1
11	1	0	0	1
10	1	1	1	0

(f)

$\hat{a}\hat{b}$ \ $\hat{c}$	0	1
00	0	--
01	1	--
11	--	0
10	1	--

(g)

$\hat{a}\hat{b}$ \ $\hat{c}$	0	1
00	0	0
01	1	--
11	--	0
10	1	--

$Sum = \hat{a} \oplus \hat{b} = a \oplus b$

Figure 11.2.1: (c) KMaps for sum and carry of the half-adder, (d) creation of intermediate variables $\hat{a},\hat{b},\hat{c}$ *, (e) Sum as function of intermediate variables* $\hat{a},\hat{b},\hat{c}$ *created in step d, (f) The incompletely specified KMap for SUM (*$\hat{a},\hat{b},\hat{c}$*), (g) Realization of SUM from incomplete specification.*

Based on the full-adder and half adder, we can build by hand a block scheme of an adder that adds three 2-bit numbers as shown in Figure 11.2.2. This will lead to a circuit with k ancilla bits. The full adder circuit can be realized also directly from KMap by any reversible synthesis method that converts a non-reversible function to a

reversible function. In Figure 11.2.2 the ancilla bits initialized to 0 (as in Figure 11.2.1a) are not shown. It is easy to rewrite Figure 11.2.2 to a quantum array (Figure 11.2.2b), so from now on we will not show details of lower stages of design for adder blocks of quantum oracles.

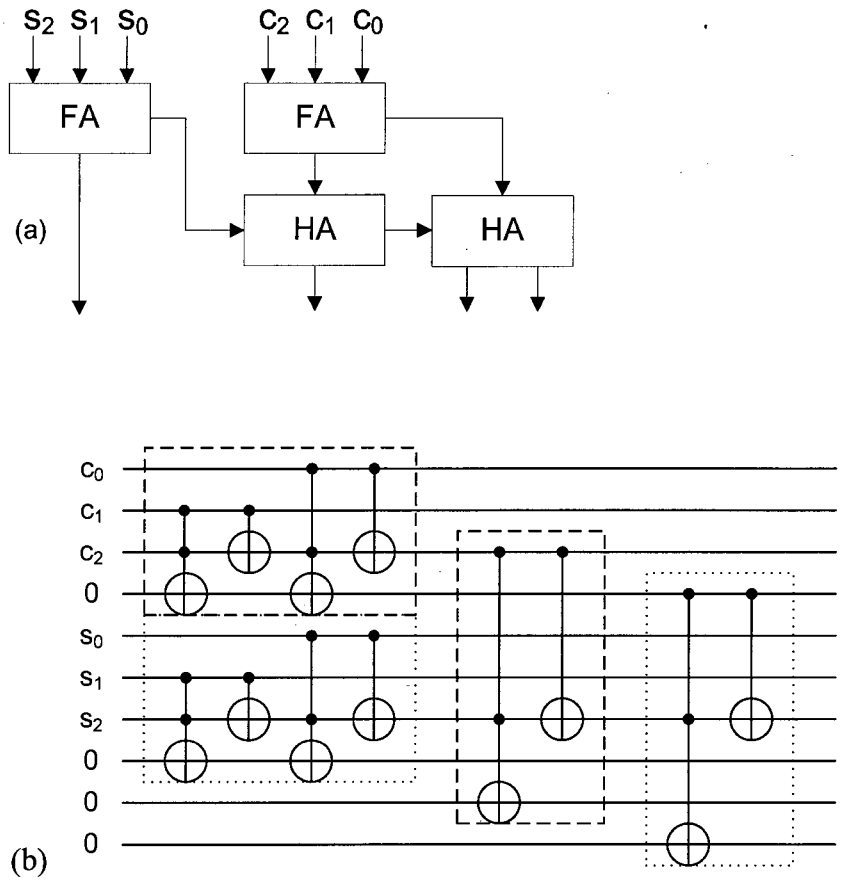


Figure 11.2.2 Block diagram of an adder of three 2-bit numbers. (a) a block diagram, (b) a corresponding quantum array.

Design of quantum adders is a well-developed area of research with relatively many publications available [Khan05a, Li06]. We will however not go deeper into adders design in this dissertation as our concern is on one hand on the automated design of

general-purpose reversible functions from irreversible specifications and on the other hand, on the specific methods for the design of oracles.

11.3. "COUNT ONES" Circuit.

The "Count Ones" circuit is one of the most useful circuits in oracle building. It will appear in many oracles from this dissertation, and also in some other oracles that I built but that are not included into the thesis. This circuit is always a part of the cost function calculation block of the oracle. It occurs in nearly all oracles that we have already built for problems in which the solution cost is being optimized. This circuit is useful in every case when one wants to calculate the number of bits "one" in a binary vector, it is thus used in many cost function blocks being parts of oracles. There are many methods to design this block. The simplest method is to create a specification of the block as a truth table, KMap or netlist and next use respective software from previous chapters or other software that converts irreversible logic specification to reversible logic and next designs a quantum array. Another approach is hierarchical. The Hierarchical design style we found useful in many cases while designing oracles. In the hierarchical approach the circuit is composed from smaller blocks, as shown in Figure 11.3.1. Finally, the "Count Ones" quantum array is shown in Figure 11.7.2 (left up). The circuit has two-levels. The first level consists of two full-adders and one half-adder which add 8-bit number in parallel and forms three 2-bit numbers. The second

level is an adder that adds the three 2-bit values which are the result of the previous adder.

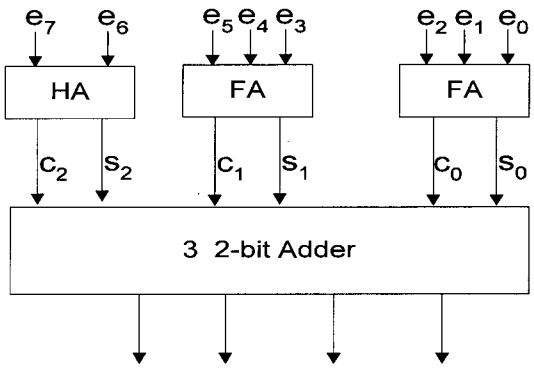


Figure 11.3.1: Block "Count Ones" realized using binary Half-Adders and Full-Adders. The block at the bottom is the adder from Figure 10.2.2. Ancilla qubits are not shown.

The design approach simplifies the circuit design process and the number and complexity of gates, at the price of increasing the number of ancilla qubits.

Now we will show a systematic method for deriving the "Count Ones" circuit for three inputs. The point here is not only to derive this particular circuit but to show a method to synthesize any circuit specified as a truth table. In theory, this method can be used for any binary circuit presented in this thesis.

In order to derive the circuit for the "Count Ones" block, the first thing to do would be to draw the Karnaugh map. The design here is assuming that there are at most three "ones" in order to make the circuit simpler and more cost-effective. The better designs of "Count Ones" circuit were presented earlier, but the method here is the most

general as it is based on a tabular representation of the function (Figure 11.3.2) and uses the general purpose AND\EXOR synthesis.

ab \ c	0	1
00	0	1
01	1	2
11	2	3
10	1	2

Figure 11.3.2: Karnaugh map for "Count Ones" circuit without binary encoding. The number in each cell corresponds to the number of input values "1" in its argument.

For instance, in the case of the graph coloring oracle, every variable a, b, c is 1 if it is representing a color. It is 0 otherwise. The circuit calculates the number of ones in the input. Figure 11.3.3 is the most basic form that our notation can take. It simply takes all possible values of the three "colors", and then has the amount of one's in each value listed in each cell. In order to convert the Karnaugh map into a form that can be used to design the circuit, we must convert it to a binary encoding (Figure 11.3.3).

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

O_1O_2

Figure 11.3.3: Karnaugh map for "Count Ones" obtained from Figure 11.3.2 after binary encoding.

The cells in the Karnaugh map have two-bit values within them, as there are up to three ones (colors). Again, we must separate this KMap into two one-bit Karnaugh

maps to derive the circuit for each output bit O_1 and O_2. The separated KMaps are shown in Figures 11.3.4 and 11.3.5.

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

O_1

Figure 11.3.4: Karnaugh map for "Count Ones" qubit O_1.

The product groups in Figure 11.3.4 for an AND/EXOR circuit is:

$$O_1 = ab \oplus bc \oplus ac$$
$$= b(a \oplus c) \oplus ac$$

This equation is our familiar majority function equation for which we found very efficient solutions using CV/CV†/CNOT gates in previous chapters, especially in chapter 7.

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

Figure 11.3.5: Karnaugh map for "Count Ones" qubit O_2.

The product groups in Figure 11.3.5 are

$$O_2 = \overline{a}\overline{b}\,c \oplus \overline{a}\,b\overline{c} \oplus abc \oplus a\,b\overline{c}$$
$$= \overline{a}(c \oplus b) \oplus a(\overline{c \oplus b})$$
$$= a \oplus b \oplus c$$

This is a linear circuit, thus it is also inexpensive. The combined logic functions from Figure 11.3.4 and 11.3.5 each form their own quantum circuit, illustrated in Figure 11.3.6. We can observe that it is the familiar Quantum Adder that we have already synthesized (more optimally) in Figure 11.2.1b.

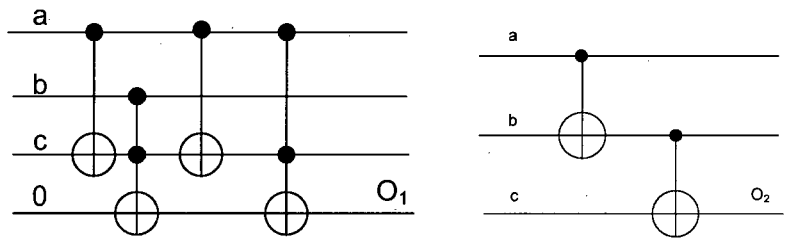


Figure 11.3.6: Separate Quantum Arrays for the "Count Ones" circuit from Figure 11.3.3.

11.4. Binary Equality, Inequality and Order Comparators.

Figure 11.4.1 is the Karnaugh map (*inverted*) of the binary equality comparator that finds many applications as the decision block in oracles.

a_0a_1 \ b_0b_1	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

Figure 11.4.1: Inverted Karnaugh map of the C block, the binary equality/inequality comparator. This KMap realizes the Equality Comparator. Its negation realizes the Inequality Comparator.

The function realized by the KMap from Figure 11.4.1 can be written as follows:

$$
\begin{aligned}
M &= \overline{a}_0\,\overline{a}_1\,\overline{b}_0\,\overline{b}_1 \oplus \overline{a}_0\,\overline{a}_1\,\overline{b}_0\,b_1 \oplus a_0\,a_1\,b_0\,b_1 \oplus a_0\,\overline{a}_1\,b_0\,b_1 \\
&= \overline{a}_0\,\overline{b}_0\,(\overline{a}_1\,\overline{b}_1 \oplus a_1\,b_1) \oplus a_0\,b_0\,(\overline{a}_1\,\overline{b}_1 \oplus a_1\,b_1) \\
&= (\overline{a}_1\,\overline{b}_1 \oplus a_1\,b_1)(\overline{a}_0\,\overline{b}_0 \oplus a_0\,b_0) \\
&= (a_1 \,\Theta\, b_1)(a_0 \,\Theta\, b_0) \\
&= \overline{a_1 \oplus b_1} \bullet \overline{a_0 \oplus b_0}
\end{aligned}
$$

Where Θ denotes the equivalence operator (equivalence is a negation of EXOR).

Figure 11.4.2 is the classical schematic of this comparator for two bits in each of the compared inputs. It uses 2 EXNOR gates (equivalence gates) leading to a NAND gate (as the KMap in Figure 11.4.1 was inverted). Extension of this circuit for arbitrary length n of words A = (a_0, a_1, …, a_n) and B = (b_0, b_1,…, b_n) is trivial and can be found using our synthesis methods or by hand as a bit-by-bit extension of the circuit from Figure 11.4.2.

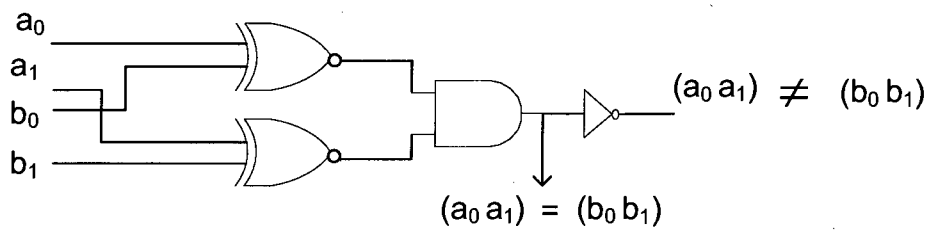


Figure 11.4.2: Classical representation of the equality/inequality comparator (C block) for two 2-qubit words.

Now we use the Toffoli gate to implement the circuit, creating the quantum comparator (Figure 11.4.3). The Toffoli is a universal gate, and can be used to represent any basic classical function in a reversible manner. Also note the presence of the EXOR function (in Feynman gate) and the NOT gates (the inverters). Observe the NAND realized by initializing the bottom qubit in Figure 11.4.3 to the value of "1".

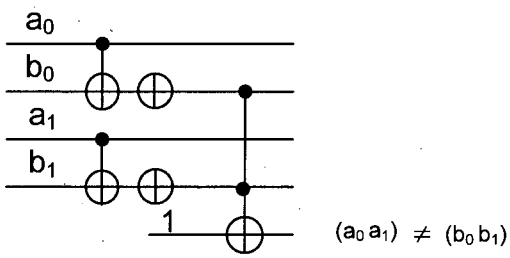


Figure 11.4.3: Quantum Inequality Comparator (C Block) for 2 qubits in a word using one ancilla qubit. This circuit can be easily extended to any length of words (a_0 .... a_n) and (b_0 .... b_n). Inputs b_0 and b_1 are not restored at outputs as this circuit is not an oracle.

Observation.

The comparator finds, for instance, application in graph coloring. All the graph's adjacent node color encodings should go through such comparators. Since each node represents a country then any nodes connected to it (adjacent countries) should not use

the same color. The comparator is designed to test whether any single node-node connection violates the coloring rules. If it does, it will come out logic 0, as can be inferred from the Karnaugh map. This 0 will affect the other comparators' result since the output from every comparator is ANDed together to provide the answer for the question "does this entire graph coloring configuration obey the coloring rules?"

Comparators are very useful blocks for many other oracles as well. There are many types of comparators that calculate values of various relations such as $>, \geq, \leq$ or $\neq$, but in this section I was first concerned with the simple equality (=) comparator which finds most applications in oracles. We designed binary and multiple-valued-input comparators of various types for quantum oracles. Here we will build the "Greater Than" Comparator using the hierarchical equation method. We first derive the Boolean function for comparator that compares two 4-bit numbers ($S = s_3 s_2 s_1 s_0$ and $B = b_3 b_2 b_1 b_0$). We first compare the Most Significant Bit (MSB), if s_3 and b_3 has different value ($s_3 \oplus b_3 = 1$) and $b_3 = 1$, then we know $B > S$. If s_3 and b_3 are the same ($s_3 \oplus b_3 = 0$), then we need to move to the next significant bit. This can be carried out until the Least Significant Bit (MSB) is reached. Based on that, we can write the Boolean function as follows:

$$out = (s_3 \oplus b_3)b_3 \oplus \overline{(s_3 \oplus b_3)}(s_2 \oplus b_2)b_2 \oplus$$
$$\overline{(s_3 \oplus b_3)}\,\overline{(s_2 \oplus b_2)}(s_1 \oplus b_1)b_1 \oplus \overline{(s_3 \oplus b_3)} \bullet$$
$$\overline{(s_2 \oplus b_2)}\,\overline{(s_1 \oplus b_1)}(s_0 \oplus b_0)b_0$$

Based on the above formula, we can directly draw the quantum array presented in Figure 11.4.4. The equations are given in some points of the circuit to help the reader analyse this design. Observe that the number of ancilla qubits was reduced to just one. This design, extended to more qubits, is used in many oracles to compare costs of potential solutions with bounds (threshold values). For instance, it is used in Graph Coloring Optimizing Oracle in Chapter 13. Observe that another method to synthesize the circuit would be based on automated design starting from a truth table.

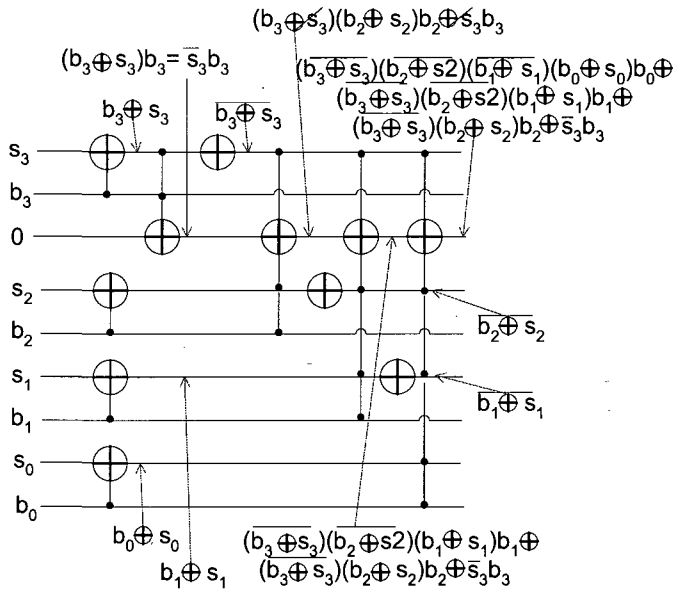


Figure 11.4.4: Binary Implementation of Quantum Comparator for 2 words of length 4. Please observe the Toffoli gate with 5 inputs in AND. This 6×6 Toffoli gate is expensive and its internal realization is not shown in Figure 11.4.4. This design requires only one ancilla qubit.

11.5. Ternary Adder and its Use in the "COUNT ONES' Circuit.

I showed above some binary realizations of few arithmetic/logic circuits. Interestingly, arithmetic design often simplifies when using multiple-valued or multiple-valued-input binary-output hybrid circuits. Below we will discuss ternary realizations. Ternary quantum logic is now the most discussed among MV logics in quantum circuits, but it is only because this area of research is so new. In [Khan05a] Khan and Perkowski invented a ternary full-adder TA, shown below in Figure 11.5.1. Observe that the symbol that was used for Feynman gate in binary quantum logic is used now for ternary Feynman gate which implements GF(3) rather than GF(2) addition.

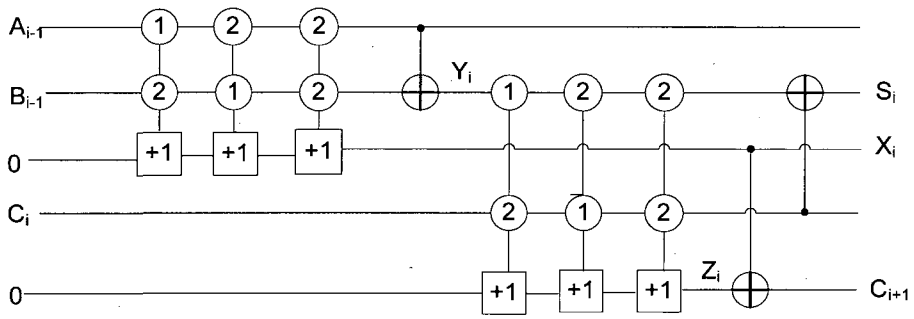


Figure 11.5.1: The ternary full-adder TA invented by Khan and Perkowski [Khan05a].

Observe that this is a hand design created by two experienced designers and researchers in the field. My formalized method from chapter 10 (section 10.2.5) found exactly the same solution. The question is, can this circuit be improved? We are not solving this problem here, instead, we use their design TA. Based on the Ternary full-adder TA, we can implement the "Count One", the number of ones circuit, as in

Figure 11.5.2. In Figure 11.5.2, the TA block stands for the ternary adder from Figure 11.5.1. Since the first from left TA block has only two inputs, its carry output is always zero. A 2-digit output s_0s_1 is enough for the result. The detailed analysis of the TA circuit is shown in Figures 11.5.3, 11.5.4, 11.5.5 and 11.5.6.

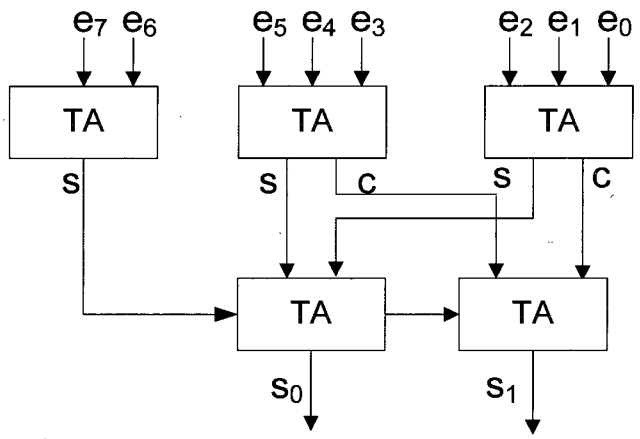


Figure 11.5.2: Block diagram of the Ternary Implementation of the "Count Ones"circuit. Ancilla qubits are not shown. The contents of blocks TA is shown in Figure 11.5.1.

B_{i-1} C_i \ A_i	0	1	2
00	0	1	2
01	1	2	0
02	2	0	1
10	1	2	0
11	2	0	1
12	0	1	2
20	2	0	1
21	0	1	2
22	1	2	0

S_i

Figure 11.5.3: The Ternary KMaps for output Si of the ternary adder TA from Figure 11.5.1.

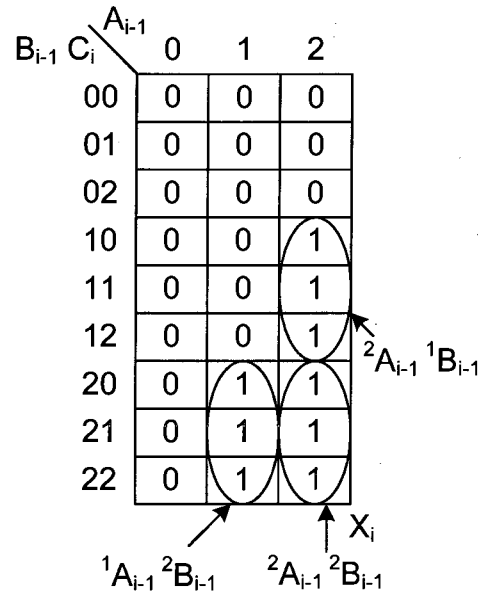


Figure 11.5.4: Ternary KMaps of signal X_i from Figure 11.5.1.

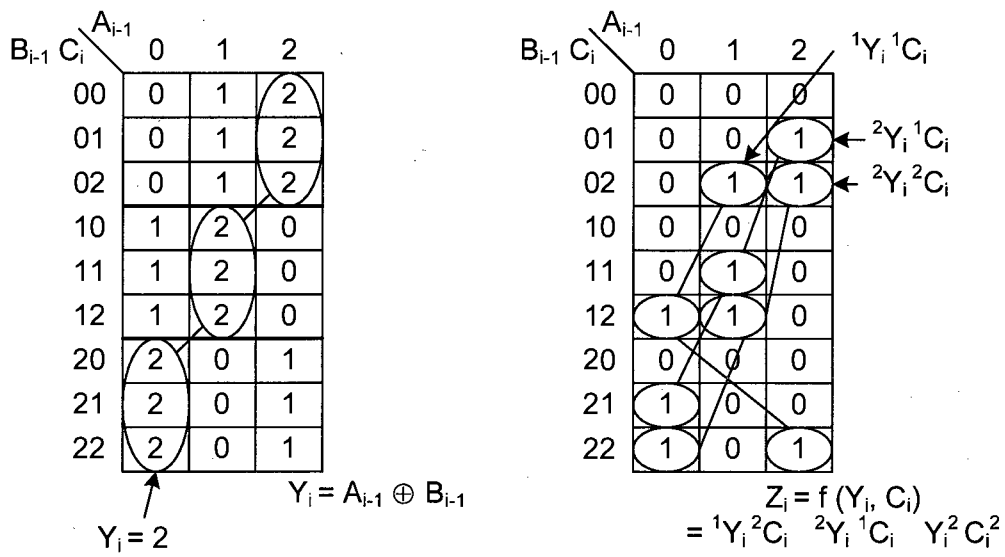


Figure 11.5.5: Ternary KMaps for signals Y_i and Z_i from Figure 11.5.1.

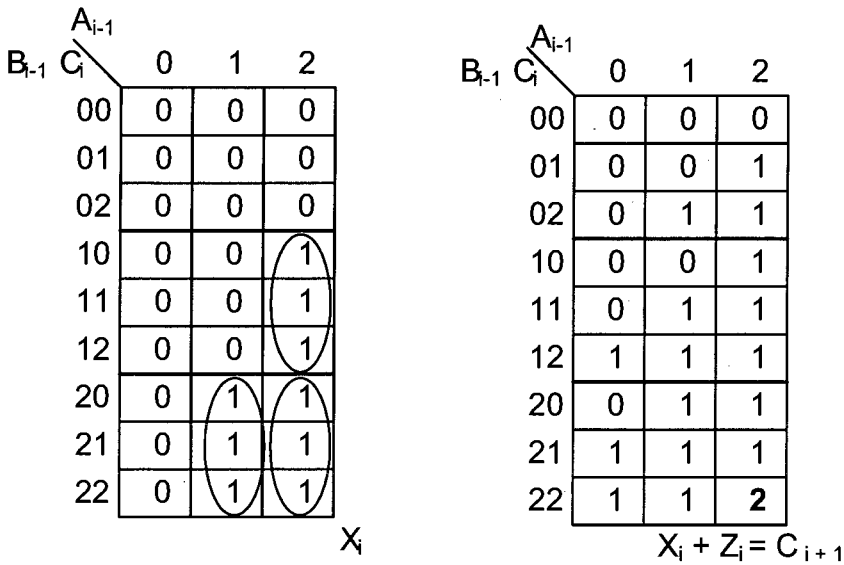


Figure 11.5.6: Ternary KMaps for signals X_i and C_{i+1} from Figure 11.5.1.

11.6. Ternary Logic "GREATER THAN" Comparator.

Ternary logic comparator is conceptually more complex than the binary logic comparator. But it is more powerful. For instance, the similar ternary equivalent inequality comparator allows to improve the color comparison in graph coloring by reducing the block's complexity. The additional advantage of such circuits is that I can implement them in the way similar to the binary comparator. Such generalizations are very useful when creating logic synthesis algorithms and possibly our synthesis methods from previous chapters can be in future extended to general-purpose multiple-valued-input multiple-valued output logic.

Applying the ternary logic, the 2-digit comparator is sufficient for my circuit design. Below I will thus focus on the 2-digit ternary comparator ($S = s_1 s_0$ and $B = b_1 b_0$). First we compare the Most Significant Digits. There are three possible assignments to make $B > S$. They are $b_1 = 2$ and $s_1 = 1$, $b_1 = 2$ and $s_1 = 0$, or $b_1 = 1$ and $s_1 = 0$. If $b_1 = s_1$, then we need to compare b_0 and s_0 with the similar manner. Based on the above ideas, the 2-digit ternary "Greater Than" Comparator can be designed, see Figure 11.6.1. Observe that this circuit is in essence hybrid as qubits $b_1 s_1$ b_0 and s_0 are ternary and qubit out is binary.

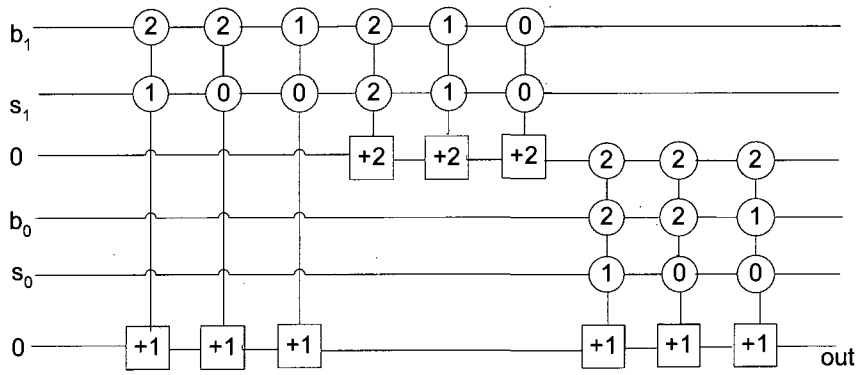


Figure 11.6.1: The Ternary Implementation of "Greater Than" Comparator.

Analysis of this circuit can be performed in a very similar way to the circuit from Figure 11.5.1. Again, this quantum array was built on hierarchical reasoning but can be also designed from hybrid truth table specification using automated tool.

11.7. The Binary Compressor Tree.

The binary Compressor Tree idea is used to generate the "Count Ones" circuit from section 11.3 using a more powerful synthesis method. There are two tasks to be accomplished by the "Compressor Tree" block:

1) To count the number of ones in the input data (this can be for instance the number of non-zero spectral coefficients in the FPRM minimization problem (see Chapter 15).

2) To compare the number of ones with the threshold value. If the number of ones in the coefficients is less than the threshold value, the circuit will output a one, otherwise the circuit produces a zero.

The "Count Ones" function can be accomplished using a tree of half-adders and full-adders and is also known in the arithmetic design community as the 10:4 compressor. The quantum implementation for the half-adder is shown in Figure 11.2.1a and the full-adder is shown in Figure 11.2.1b. The 8:4 compressor based on a tree of full-adders and half adders is shown in Figure 11.7.1. The compressor circuit includes two levels, with the first level consisting of two full-adders and one half-adder which add two 3-bit values and one 2-bit value parallel and form three 2-bit numbers. The second level is an adder tree that compresses the 6-bit value from the first stage into a 4-bit value. The detailed implementation as a quantum array is shown in Figure 11.7.2 (in

the leftmost box). We see here that the result $S_0S_1S_2S_3$ along with the threshold value $b_0b_1b_2b_3$ serve as the inputs to the comparator.

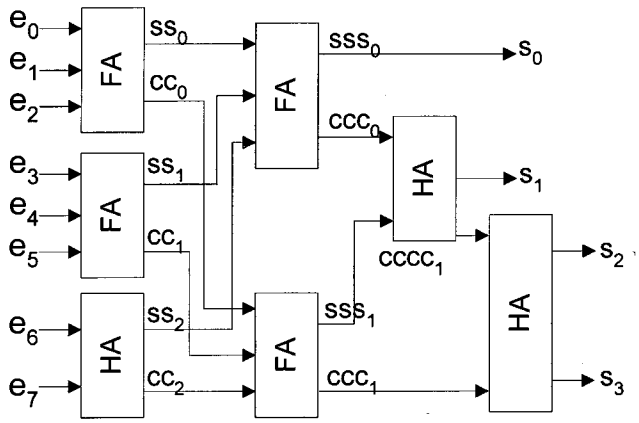


Figure 11.7.1: Block diagram of the 8:4 Compressor Tree. The binary FA and HA adder blocks were explained in section 11.2.

We build the comparator as in section 11.4, we first derive the Boolean function for a comparator that compares two unsigned 4-bit numbers ($S = s_3s_2s_1s_0$ and $B = b_3b_2b_1b_0$). We first compare the Most Significant bit (MSB), if s_3 and b_3 have different value ($s_3 \oplus b_3 = 1$) and $b_3 = 1$, then we know $B > S$. If s_3 and b_3 are the same ($s_3 \oplus b_3 = 0$), then we check the next significant digit. This can be carried out until the Least Significant bit (LSB) is reached. Based on that, we can write the Boolean function as the following:

$$out = (s_3 \oplus b_3)b_3 \oplus \overline{(s_3 \oplus b_3)}(s_2 \oplus b_2)b_2 \oplus \overline{(s_3 \oplus b_3)}\,\overline{(s_2 \oplus b_2)} \bullet$$
$$(s_1 \oplus b_1)b_1 \oplus \overline{(s_3 \oplus b_3)} \bullet \overline{(s_2 \oplus b_2)}\,\overline{(s_1 \oplus b_1)}(s_0 \oplus b_0)b_0$$

Based on the formula derived there, we design the quantum circuit as shown in Figure 11.7.2 (in the rightmost box at the bottom). The inverse circuits are just mirror reflections of their basic circuits, and thus the inverse butterfly is drawn by mirroring in inverse order all gates from the butterfly. This is because in binary reversible logic the generalized Toffoli gates, Toffoli gates, Feynman gates, Fredkin gates and NOT gates are their own inverses.

Gates are not their own inverses in the ternary logic, but designing inverse circuits is also straightforward in ternary logic [Khan05a]. Let us observe that this circuit has 8 ancilla bits. In theory the number of ancilla bits can be reduced to one.

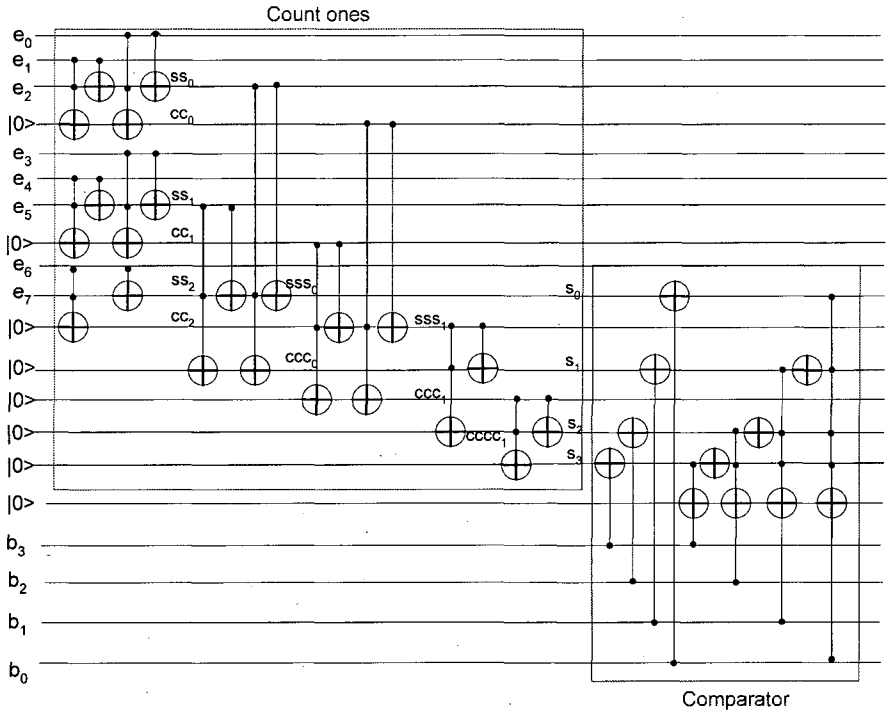


Figure 11.7.2: Binary Quantum Array for the 8:4 Compressor from Figure 11.7.1 and Comparator for 8-bit data (e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7) and 4 bit data (b_0, b_1, b_2, b_3).

11.8. Multiple-Valued Logic Realization of the Compressor Tree.

As described in section 11.7, the (binary) "Compressor Tree" circuit consists of a compressor and a comparator. The compressor will have a delay and cost proportional to the number of stages in the compressor tree. In the small 3-variable example described previously, 3:2 and 2:2 binary compressors are used. For larger functions, the compressor tree can be optimized by using larger compressors and compressors that are based on the signed binary digit set (a kind of multiple-valued logic system).

The signed binary number system still uses a radix-value of 2 but allows for a digit set of $\{\overline{1},0,1\}$, where $\overline{1}$ represents the value of -1 (a negative unity). This is redundant in that some values may be expressed with two-different digit strings (eg. $+1 = 01 = 1\overline{1}$). Efficient compressors may be designed using the signed binary adder as a component. Because three distinct digits are used, it is convenient to implement the signed digit adder as a quantum-ternary-valued circuit. We also note that these types of adders have the desirable property of constant delay regardless of the word-length and can be used as the basis for other high-speed arithmetic circuits. Table 11.8.1 first appeared in [Harata87] and illustrates how the redundant digit set is exploited to prevent long carry ripples.

SIGNED BINARY ADDITION TABLE USED TO PREVENT LONG CARRY PROPAGATION CHAINS

Addend + Augend Digits in Position i $A_i B_i$	Sign Information of of Digits in Position $i-1$ Precondition $A_{i-1} B_{i-1}$	Intermediate Carry Digit, c_{i+1}	Intermediate Sum Digit, s_i
$\overline{1}+\overline{1}$	Not Used	$\overline{1}$	0
$\overline{1}+0$	Either is Negative	$\overline{1}$	1
$\overline{1}+0$	Neither is Negative	0	$\overline{1}$
$0+0$	Not Used	0	0
$1+\overline{1}$	Not Used	0	0
$1+0$	Either is Negative	0	1
$1+0$	Neither is Negative	1	$\overline{1}$
$1+1$	Not Used	1	0

*Table 11.8.1: Signed Binary table used to prevent long carry propagation chain. The values s_1 and c_1 occur when **either** of the digits at next position to the right are negative while s_2 and c_2 occur when **neither** of the digits to the right are negative. The portion of the circuit that performs this computation is called the pre-condition.*

Our implementation of this adder as a ternary-quantum circuit is shown in Figure 11.8.1 where we use the following encoding scheme: $0 \leftrightarrow 0$, $1 \leftrightarrow 1$, $\overline{1} \leftrightarrow 2$. To build the signed-digit adder, logic to differentiate between s_1 and s_2 is necessary. The subcircuit for computing the pre-condition that "either input is negative" is based on the ternary GF Toffoli gates as shown the in upper-left box in Figure 11. 8.1.

Based on above Table 11.8.1, the sum and carry values are:

$${}^0A + {}^0B + {}^0A{}^0B = {}^2S_2$$

$$2 \bullet s_2 = s_1$$

$$C_{i+1} = S(2 \bullet (A_i \oplus B_i)) = S \oplus (2 \bullet (A_i \oplus B_i))$$

where $\oplus, \bullet$ are GF(3) addition and multiplication, respectively. ${}^0A$, ${}^0B$ are post literals.

From the formulas, the entire adder can be implemented as shown in Figure 11.8.1. The lower-left box shows the implementation for the sum digit and the lower-right box indicates the portion for the for the carry digit. This circuit can be formally obtained using methods from chapter 10.

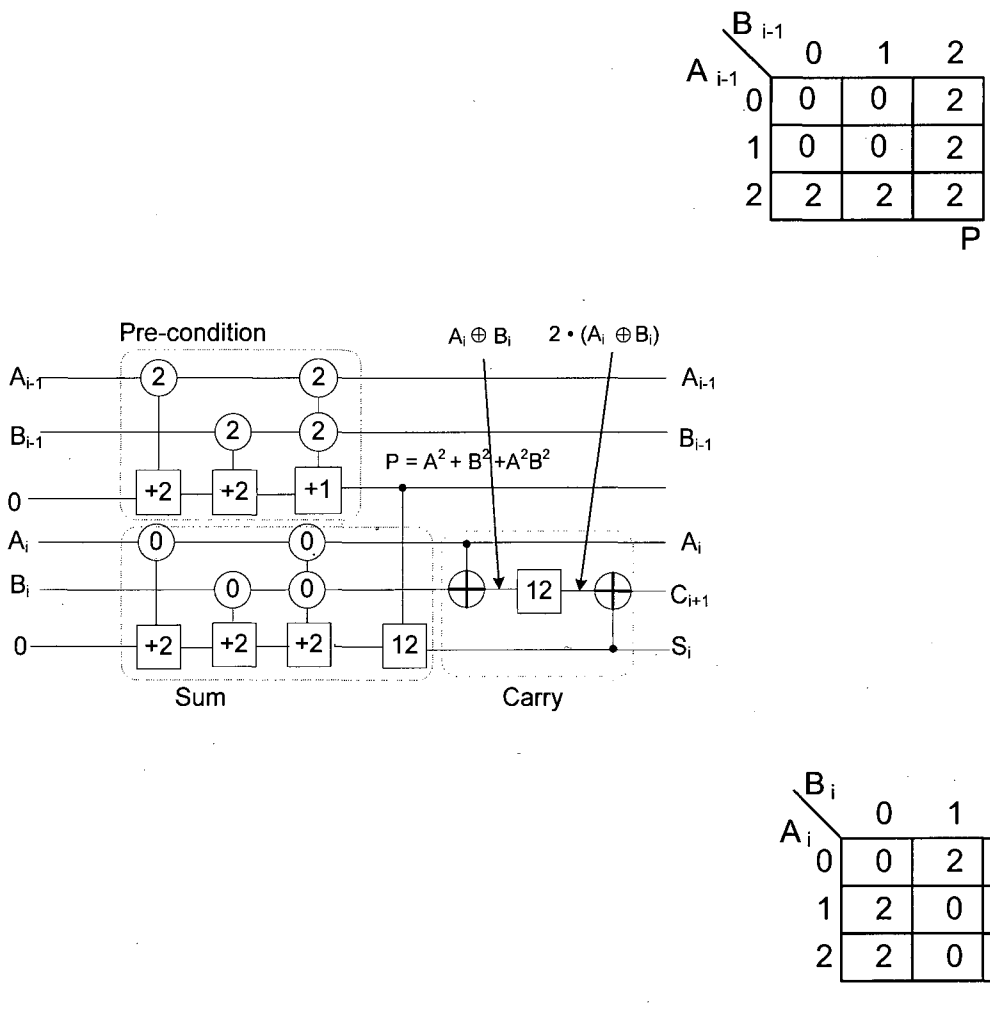


Figure 11.8.1: Quantum Array for the Ternary Sign Adder Circuit.

Observe the use of two ternary Feynman gates that realize modulo-3 addition. Analysis of this circuit can be done similarly as of the circuit from Figure 11.5.1.

11.9. The Sorting / Absorbing Circuit.

The sorting/absorbing block exists in oracles when there is a need to convert the set with repeated elements to the set with non-repeated elements. This is an iterative algorithm, which works by exposing the data to a number of butterfly iterations of SAP (sorting/absorbing processor) blocks. The simplified design to sort/absorb four 3-bit numbers is presented in Figure 11.9.1. It can be used for instance in those variants of Graph Coloring algorithms that try to find the coloring with the minimum (chromatic number) number of colors (in the so-called optimizing oracle).

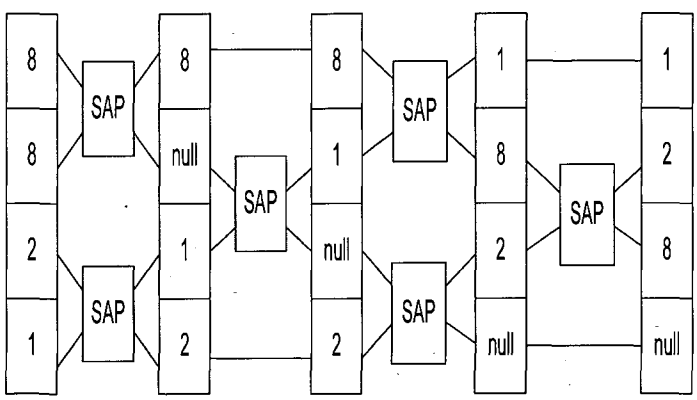


Figure 11.9.1: Butterfly iterative circuit for sorting/absorbing to be used as a block in cost optimizing oracles.

Figure 11.9.1 presents an example of the circuit (algorithm) for four numbers. Here it is acting on the unsorted set of numbers 8, 8, 2 and 1. As one can see, repeated iterations end in the set being sorted from the smallest to the largest number, with repeating entries reduced to nulls (i.e. absorbed). Figure 11.9.2 is the diagram of the single SAP block with 3-bit inputs a and b, 3-bit outputs c and d, binary outputs z and v. In multiple-valued design the qudits a and b can be of radices higher than 2.

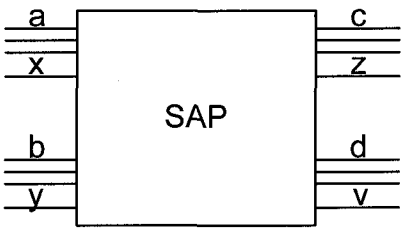


Figure 11.9.2: The symbolic schematics of the SAP processor with notation used for its inputs and outputs.

Each SAP block can sort two input numbers a and b such that the smaller one will be output from the output port c and larger one will be output from port d. If two inputs to SAP are equal, then one of them will become null/absorbed. However, here we run into a problem. In this quantum circuit, there cannot be a notation for an absorbed bit. Initially, all of the inputs in Grover algorithm are put through Hadamard gates. This means that for instance in the graph coloring problem the number of "colors" is always a power of 2. This does not affect the performance of the equality comparator; however, the Sorter/Absorber will be severely affected. Colors that should not exist (i.e. larger than worst case number of colors) would be sorted alongside colors that should exist, and thus decrease the efficiency. In order to compromise this problem, I

have added the tagged bits x, y, z and v. These tagged bits are bits that are attached to the color encodings. Those that have the tag value of 1 are colors. Those with tag 0 are colors that were absorbed in the SAP, and are considered nulls. After all color encodings pass through this "butterfly" sorter/absorber, the output will sort these color encoding from the "smallest" to the "largest" and the last qubits at the bottom will be occupied with nulls.

The SAP involves the interaction between the tag inputs and the data inputs. The circuit for sorting/absorbing the tags can be represented as a series of maps (these are not standard Karnaugh maps). From these maps, we will derive the classical notation of the circuit and then convert it to the quantum form. The first map is a general map that denotes the outputs c, d, z, v given the state of word a compared to state of word b, as well as the states of the input tag signals x, y (Figure 11.9.3). The determination of whether a=b, a>b, or a<b is defined by multiplexers, which will be shown later. We found these new maps that combine logic variables together with arithmetic predicates very useful in some oracle synthesis problems.

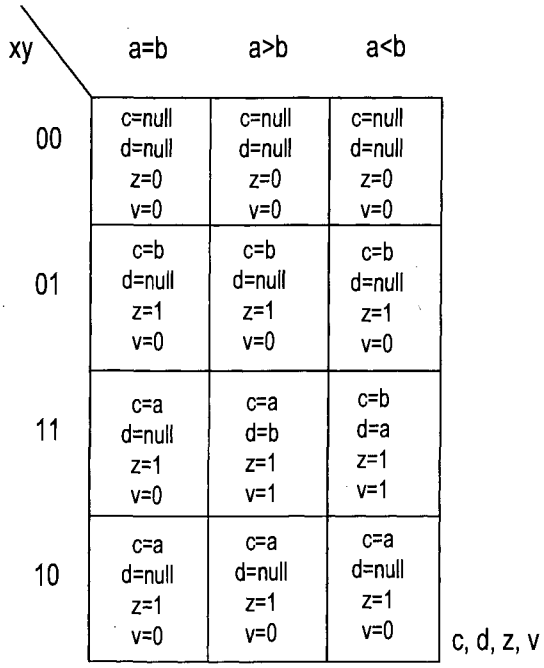


Figure 11.9.3: The map for cdzv the output signals c, d, z, v as the functions of their inputs x, y, and values of predicates (a = b), (a > b), (a < b). This quantum map generalizes the input data from bits to words a, b and is thus a new concept in synthesis.

The Figure 11.9.3 map specifies the action of the SAP. In order to derive the circuitry, we have to separate the map from Figure 11.9.3 into 4 different maps:

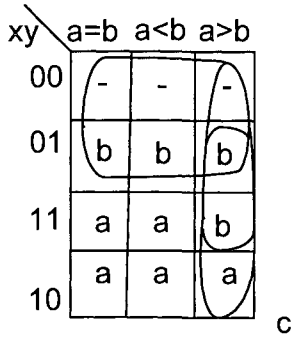


Figure 11.9.4: The KMap for c. Observe that c is in general a k-input word, not a bit. Columns (a = b), (a < b) and (a > b) are Boolean predicates with 2 k-bit arguments each. They are realized as comparators = , < , >, respectively. The design of the

"equivalence" and "Greater Than" operators was already discussed. The detailed design is presented in section 11.10.

Figure 11.9.5 below represents the classical circuit for a, b, c of k bits. Observe that c is equal to b if control of multiplexer is 1. This means that a>b and y=1 when x=0.

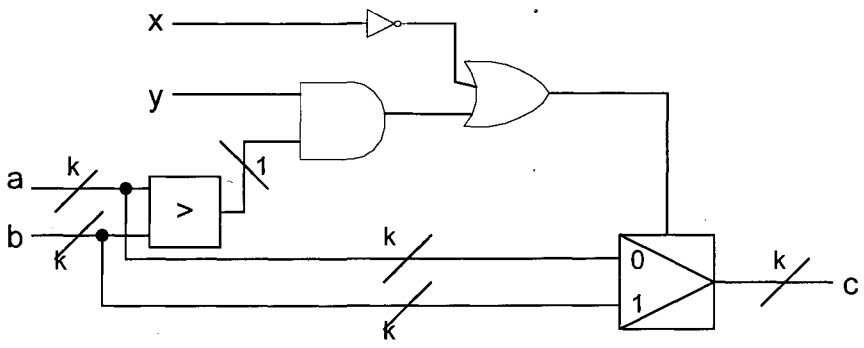


Figure 11.9.5: Classical circuit for qubit bus c of k bit-width. This circuit was calculated from the KMap in Figure 11.9.4. For k = 1 the circuit can be easily directly converted to a quantum array.

xy	a=b	a<b	a>b
00	-	-	-
01	-	-	-
11	-	b	a
10	-	-	-

d

Figure 11.9.6: The KMap for bus d of arbitrary width.

Figure 11.9.7 below presents the classical circuit for d. We assume here width k of signals a, b, c. So, strictly speaking it is not a circuit but a block diagram.

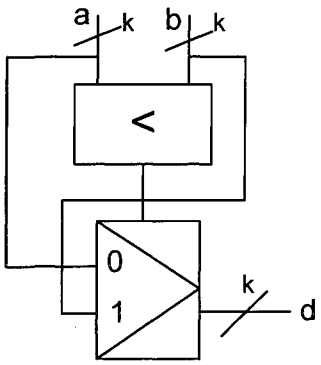


Figure 11.9.7: The classical circuit for bus d for k width of qudits in data.

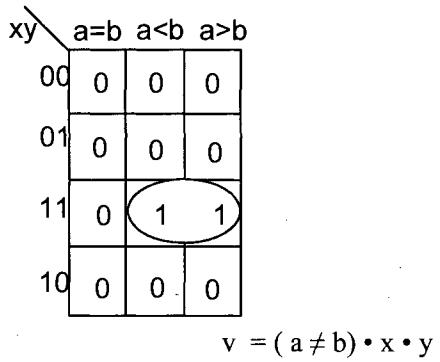


Figure 11.9.8: The KMap for v.

Figure 11.9.9 below represents the classical circuit for the tag qubit v.

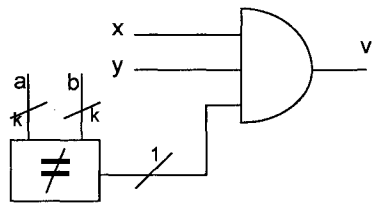


Figure 11.9.9: Classical circuit for the tag qubit v. Words a and b are of width k.

From the classical circuit, we can derive our quantum circuit for finding v. This is given in Figure 11.9.10.

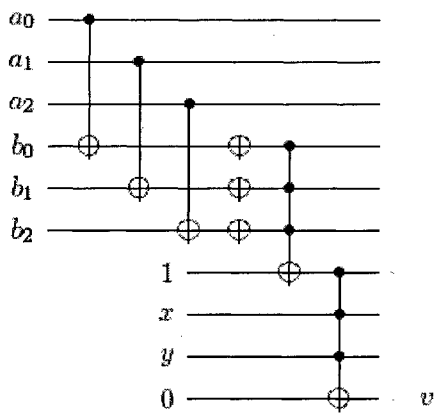


Figure 11.9.10: The quantum circuit for the tag qubit v. In this particular example words a and b have three qubits each.

xy \ z	a=b	a<b	a>b
00	0	0	0
01	1	1	1
11	1	1	1
10	1	1	1

z

Figure 11.9.11: The KMap for z.

Figure 11.9.12 below represents the classical circuit for z.

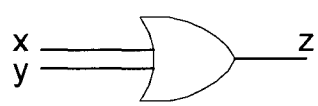


Figure 11.9.12: Classical circuit for the tag qubit z

From the classical circuit, we can derive the quantum circuit for z (Figure 11.9.13). De Morgan's Theorem was used.

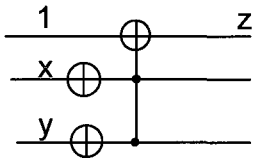


Figure 11.9.13: Quantum circuit for the tag qubit z.

What remains is to design the arbitrary comparators of width k using quantum arrays. This will be done in section 11.10.

11.10. The Iterative Comparator of A = B, A > B and A < B.

To design an iterative circuit to compare the two numbers (for instance color encodings) a and b, we can use a state machine approach. Bit streams a and b may contain k bits each (are buses of width k). By putting a_i b_i into the state diagram, depending on their value, the next state Q_1+ and Q_2+ will change. The comparison will act on the qubits of the inputs from the LSB (least significant bit) to the MSB (most significant bit). The last state of Q_1 and Q_2 represent the results of this two-bit stream comparison.

We used a Karnaugh map to define the circuit. In our case, we used 2-digit bitstring (encoding) representing a > b, b > a, and a = b. Notice that since there are only 3 different outputs, there's "too much" room when we simplify the Karnaugh map, so an entire row will be turned into "don't cares" which can be changed to suit the circuit. The encoding for a=b was 00, a > b: 10 and a < b: 01. Since 11 has no corresponding encoding value, its row is struck off as a "don't care".

We designed the structure of the Karnaugh map (pre-encoding) based on a state machine that defined how the different states a > b, a < b, and a = b are changed depending on whether the entry is 00, 01, 11, or 10. Those values are placed at the top (columns) of the Karnaugh map, and the encodings were placed as the rows. The state machine told how each of the states would react to an entry, and so we copied it down onto the Karnaugh map. We then simplified it into 2 bit encodings, and then 1-bit values (00s and 01's were considered 0's, see above for what they represent). Figure 11.10.1 presents the Finite State Machine diagram.

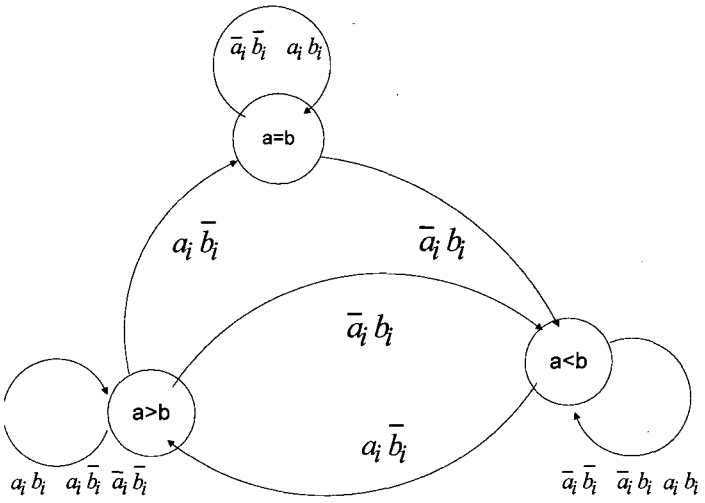


Figure 11.10.1: State machine for predicates.

We rewrite the graph from Figure 11.10.1 into KMap form, as shown in Figure 11.10.2.

Q_1Q_2 \ a_ib_i	00	01	11	10
(a=b) 00	a=b	a<b	a=b	a>b
(a<b) 01	a<b	a<b	a<b	a>b
not used - 11	---	---	---	---
(a>b) 10	a>b	a<b	a>b	a>b

$Q_1^+Q_2^+$

Figure 11.10.2: The Karnaugh map representation of the state machine graph from Figure 11.10.1.

Using the state encoding of a = b, a > b, and a < b as at the left of Figure 11.10.2, we transform the Figure 11.10.2 into the usable standard form Figure 11.10.3.

State encoding: a=b:00 a>b:10 a<b:01	a_ib_i 00	01	11	10
Q_1Q_2 (a=b) 00	00	01	00	10
(a<b) 01	01	01	01	10
not used -	---	---	---	---
(a>b)10	10	01	10	10

Figure 11.10.3: The Karnaugh map after state encoding as shown in left. We found this to be the best encoding by exhaustive search.

We can separate the map from Figure 11.10.3 to two maps to represent each output bit separately.

Figure 11.10.5 represents the Karnaugh map and the groups for ESOP minimization for Q_1^+.

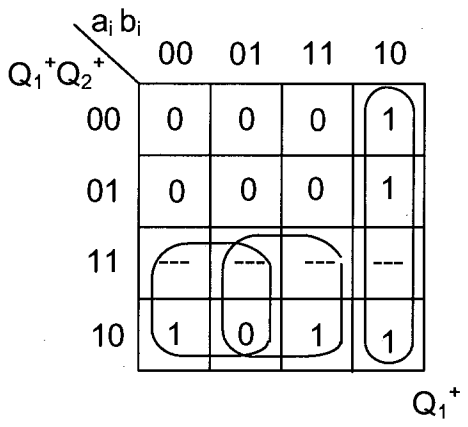


Figure 11.10.4: Karnaugh map for output Q_1^+. The groups are for ESOP synthesis.

$$Q_1^+ = Q_1\,\overline{a}_i \oplus Q_1\,b_i \oplus a_i\,\overline{b}_i = Q_1\,(\overline{a}_i \oplus b_i) \oplus a_i\,\overline{b}_i$$

Figure 11.10.5 illustrates the Karnaugh map and ESOP groups for output Q_2^+.

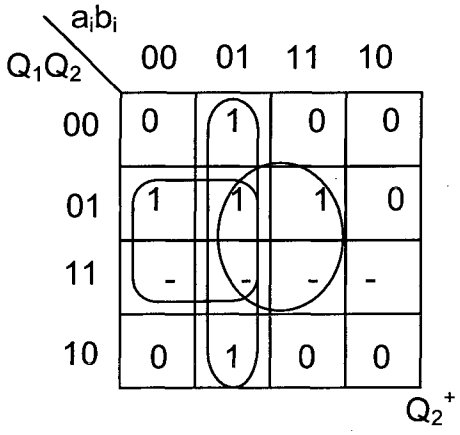


Figure 110.10.5: Karnaugh map for Q_2^+ .

From Figure 11.10.5 we obtain ESOP expression

$$Q_2^+ = Q_2 (\overline{a}_i \oplus b_i) \oplus \overline{a}_i b_i$$

Figure 11.10.6 illustrates the final quantum circuit for qubits $Q_1^+ Q_2^+$ of the iterative circuit. Observe the SWAP gates added at the right to have Q_1^+ be in the same qubit (layer) as Q_1, and Q_2^+ be in the same qubit as Q_1 .

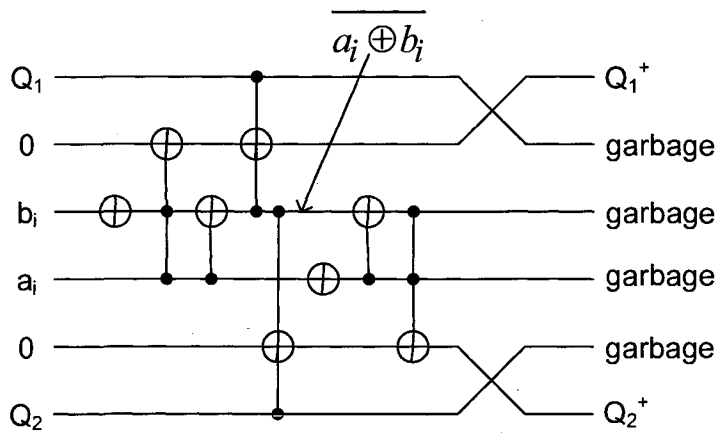


Figure 11.10.6: Circuit for $Q_1^+Q_2^+$. Please observe garbage qubits G, and the use of SWAP gates to provide the outputs Q_i^+ in the same qubit from top as the next expected qubit Q_i^+. This is a requirement of iterative circuit.

The Q_n values will continually change as the values are "read" from the least significant bit to the greatest significant bit. After no more bits remain to be read, the values of Q_n will be compared to receive the judgment of a compared to b. The iterative nature is shown in Figure 11.10.7.

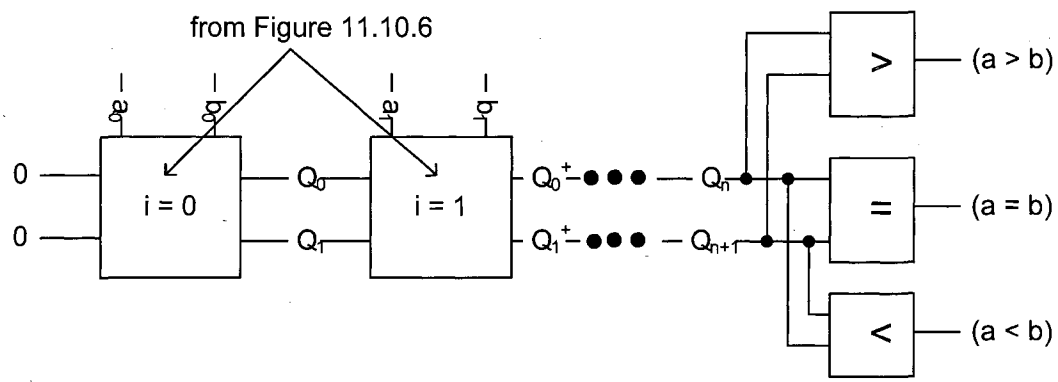


Figure 11.10.7: The iterative action of the n-bit comparator circuit. Ancilla bits not shown.

11.11. Arithmetic Reversible Blocks: Adders, Subtractors and Kernels.

It is obvious from theory in Chapter 3 that every arithmetic, counting, encoding, predicate transform or other irreversible function can be realized as a quantum circuit by adding some number of ancilla bits. The number of ancilla bits may be however excessive in some designs so we always want to find a way to reduce the number of ancilla bits. Partially it can be done by good design using automated tools. This way the number of ancilla bits can be reduced to at most m where m is the number of outputs. However, in some problems one can invent another architecture on high level, an architecture that uses reversible high level blocks.

Let us discuss one example. Suppose that I want to design a k-bit adder of two numbers as in Figure 11.11.1a. Obviously this circuit is not reversible. But I can make it reversible by repeating one of its K-word-width input words as in Figure 11.11.1b.

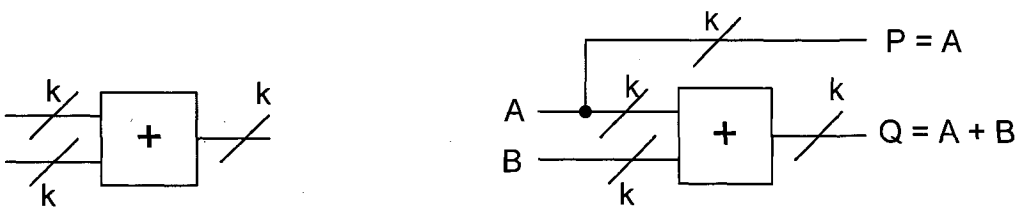


Figure 11.11.1: (a) Irreversible modulo adder, (b) the same adder made reversible by replicating its k-width input A to output. This is, in essence, the same trick as one applied to design the Feynman gate.

Using reversibility on word level we obtain the following equations.

$$\left.\begin{array}{l} P = A \\ Q = A + B \end{array}\right\} \rightarrow \begin{array}{l} A = P \\ B = Q - P \end{array}$$

Which shows that logically the inputs can be derived in a unique way from the outputs.

However in some problems it is better to have another method to achieve reversibility. For instance design from Figure 11.11.2 is better when one uses the A – B block as well.

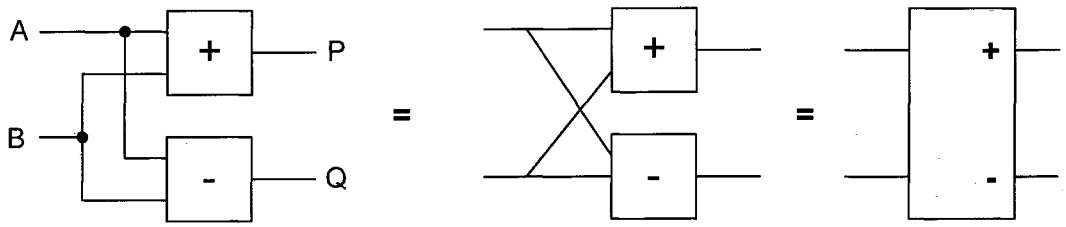


Figure 11.11.2: The reversible adder/subtractor used in Hadamard/Walsh butterflies and its notations.

This design (known as a Kernel of Walsh Transform) is used in a reversible design of Walsh Transform based on Butterflies, see Figure 11.11.4.

Before we discuss Walsh Butterfly in more detail let us observe that the Kernel block from Figure 11.11.2 is reversible, as results from solving equations in Equation 11.11.1.

$$\left.\begin{aligned} P &= A+B \\ Q &= A-B \end{aligned}\right\} \Rightarrow \left.\begin{aligned} P+Q &= 2A \\ P-Q &= 2B \end{aligned}\right\} \Rightarrow \begin{aligned} A &= \frac{P+Q}{2} \\ B &= \frac{P-Q}{2} \end{aligned}$$

Equation 11.11.1

Observe that in some technologies (such as reversible CMOS Optical and quantum) the logical reversibility of the circuit corresponds also to its physical reversibility, which means that by providing input data P, Q to outputs of the circuit we will obtain the output data A, B (as in Equation 11.11.1) at the inputs of the circuit. The role of inputs and outputs can be thus completely reserved. This circuit behavior is something entirely impossible in standard CMOS circuits as used now in VLSI.

The Walsh transform is described by a Kernel matrix $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ (we omit coefficients for simplification). By using Kronecker product (tensor product) I can build the matrix for two variables (variables corresponds to columns of kernel blocks in butterflies) and next the Butterfly Circuit using standard DSP methods [Stankovic97, Miller02, Li06]. This Butterfly circuit is shown in Figure 11.11.3.

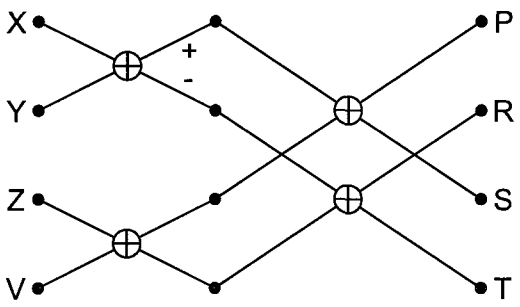


Figure 11.11.3: The butterfly of 4 kernels for 2 variables. It would require SWAP gates in quantum realization.

The schematic diagram from Figure 11.11.3 can be rewritten to the more detailed block diagram from Figure 11.11.4.

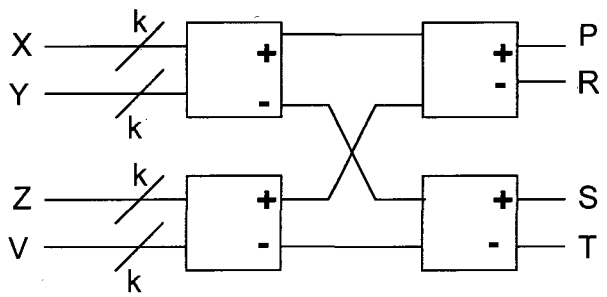


Figure 11.11.4: The butterfly from Figure 11.11.3 in another notation. This diagram shows the buses of width k, identical blocks for realization of kernels and the necessity of SWAP gates for crossing buses with width k.

Next, the circuit from Figure 11.11.4 can be rewritten to the even more detailed diagram from Figure 11.11.5 that explains the role of the adder and subtractor blocks inside the Kernels.

The Truth table of the Kernel (the + and – operations are mod_4) is given in Table 11.11.1.

x_1x_2	y_1y_2	+ mod$_4$ s_1	s_2	- mod$_4$ d_1	d_2
00	00	0	0	0	0
00	01	0	1	1	1
00	10	1	0	1	0
00	11	1	1	0	1
01	00	0	1	0	1
01	01	1	0	0	0
01	10	1	1	1	1
01	11	0	0	1	0
10	00	1	0	1	0
10	01	1	1	0	1
10	10	0	1	0	0
10	11	0	0	1	1
11	00	1	1	1	1
11	01	0	0	1	0
11	10	0	1	0	1
11	11	1	0	0	0

Table 11.11.1: The truth table of the Walsh Transform kernel for width of registers k = 2.

The quantum array for the Walsh Butterfly for 2 variables (matrix 4 × 4) is shown in Figure 11.11.5.

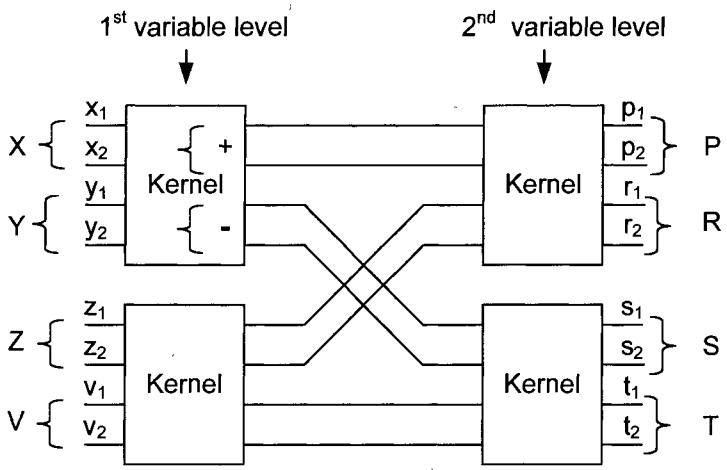
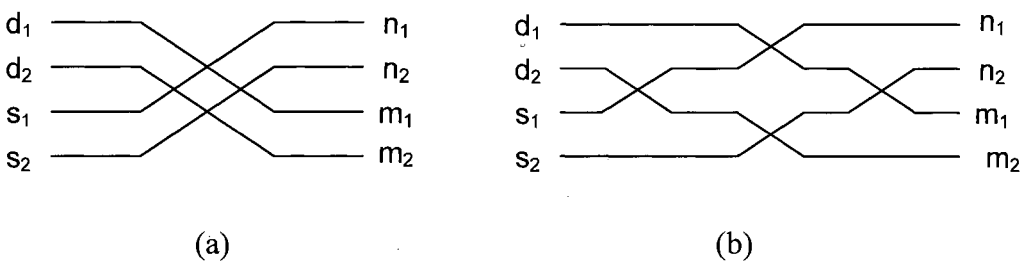


Figure 11.11.5: The quantum array for the circuit specified in Table 11.11.1 emphasizes "quantum layout" of blocks.



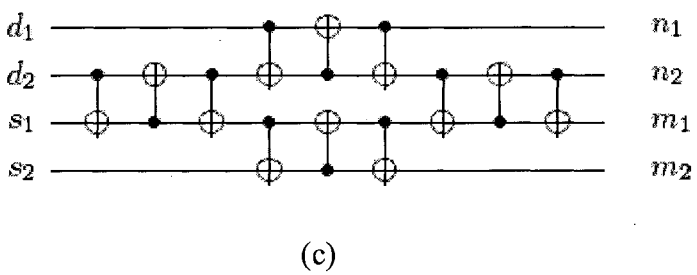
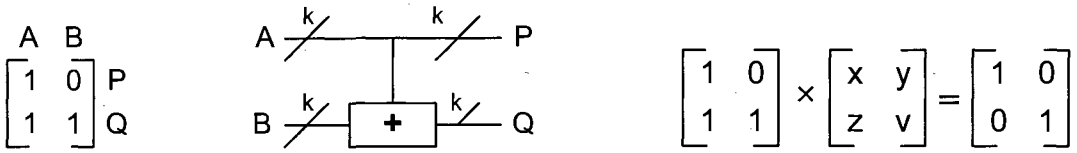


Figure 11.11.6: The detailed design of the switching network for Walsh Transform from Figure 11.11.5. (a) The symbolic switching, (b) symbolic switching rewritten to SWAP gates, (c)Quantum circuit realization of the circuit from Figure 11.11.6b using CNOT gates.

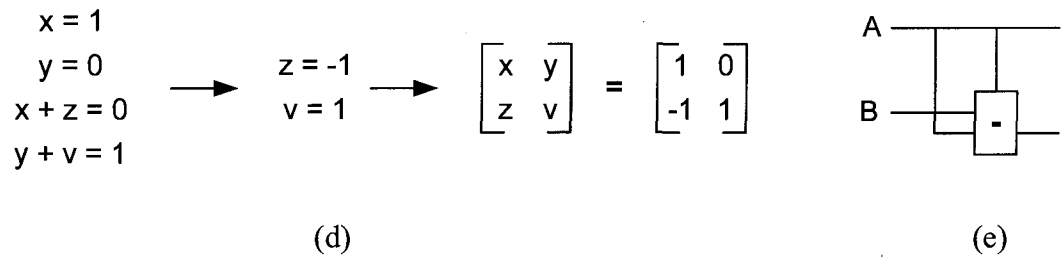
There are many spectral transforms that are based only on addition and subtraction operations. These transforms include all the family of Fixed Polarity RM, GRM etc, but also the Arithmetic Transforms used in Artificial Intelligence [Falkowski03b] and the Adding transform used in Logic Design and Image Processing [Falkowski97].

The Kernel of the Adding Transform is shown in Figure 11.11.7a. Observe that the wires are of width k. This is the generalization of the PPRM Kernel from Chapter 3. A Butterfly circuit for the Adding Transform. can be build similarly as for the Walsh (Hadamard) Transform (see Figure 11.11.7 for more details).

Finally, let us find the inverse transform to the Adding Transform. For the Kernel we obtain the equation as in Figure 11.11.7c. Solving this equation we obtain the Kernel matrix from Figure 11.11.7d which is the same as the (redrawn) Kernel of the Arithmetic Transform, Figure 11.11.7e. The reversible butterfly for this transform can be build in the same way as we have done it for the Walsh and Adding Transforms. Observe that all these circuits are perfectly reversible without any ancilla bits.



(a) (b) (c)



(d) (e)

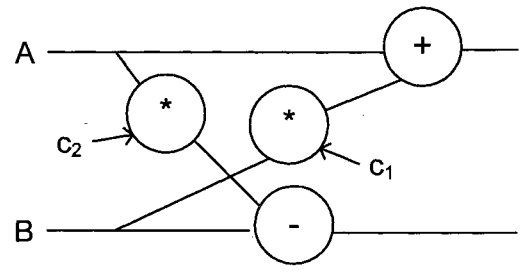
Figure 11.11.7: The reversible butterfly architectures for Adding and Arithmetic Spectral Transform. (a) The kernel of the Adding Transform, (b) The circuit for the kernel of the Adding Transform with k-bit words, (c) Matrix Equation to find the inverse Adding Transform, (d) solving the equation determines the matrix of the kernel that happens to be the kernel of the well-known Arithmetic Transform. (e) Realization of the kernel of the Arithmetic Transform with a single subtractor (it needs SWAP in quantum realization).

Observe that because Adding Transform is the inverse of Arithmetic Transform, the same circuit can be used for both, just by providing the data either at one end or the another. Again, this is not possible in classical CMOS or any known standard technology.

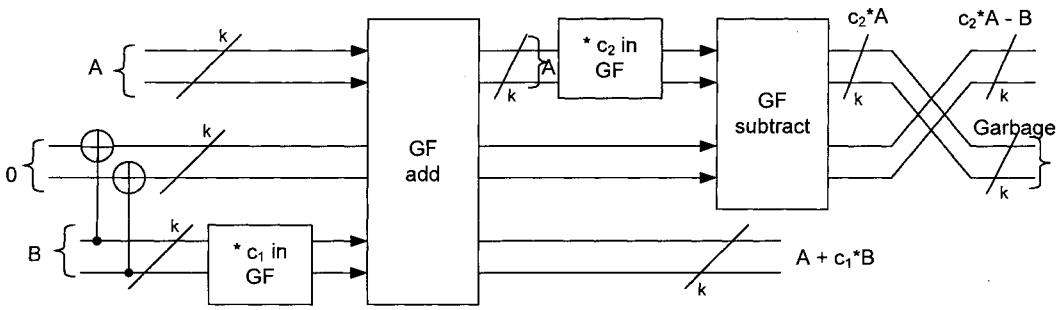
11.12. Circuits for other Spectral Transforms

Other known transforms include Fourier Transform, Haar Transform and Hough Transform. There is much published on Quantum Fourier as it is the fundament of the Shor algorithm for quantum factoring. It is also much published on Haar Transform which is the simplest Quantum Wavelet. We did not find anything on Quantum Hough Transform but this subject is very complicated, so we will drop it here. However, we would like to show the reversibility of some operations that can be used to design

various kinds of known and new spectral transforms. The general Kernel pattern for all Generalized Transforms is presented in Figure 11.12.1.



(a)



(b)

Figure 11.12.1: The Generalized Transform Kernel for Butterflies: (a) The schematic with 2 multipliers, an adder and a subtractor, (b) The quantum array on block level emphasizes hierarchical design with reversible blocks and the role of SWAP gates for quantum buses.

When coefficient c_1 and c_2 are constants, and operations +, - and * are done in a Galois Field algebra, then the generalized Kernel can also rewritten directly to a quantum array, as shown in Figure 11.12.1b. This result is not known from literature, although it is obvious. It is interesting what may be some practical applications of this fact. Using this design we can design a quantum oracle to find some certain transforms from the families of transforms; this would be a generalization of the paper by Lin,

Thorton and Perkowski [Li06]. As seen, the generalized transform from Figure 10.12.1 is a generalization of both Kernels corresponding to positive Davio and negative Davio expansions from [Li06] and Chapter 15. Similar architectures have application in Adaptive Filtering for Image Processing and Array Signal Processing.

The circuit from Figure 11.12.1b can be redrawn to the circuit from Figure 11.12.2 by adding SWAP gates.

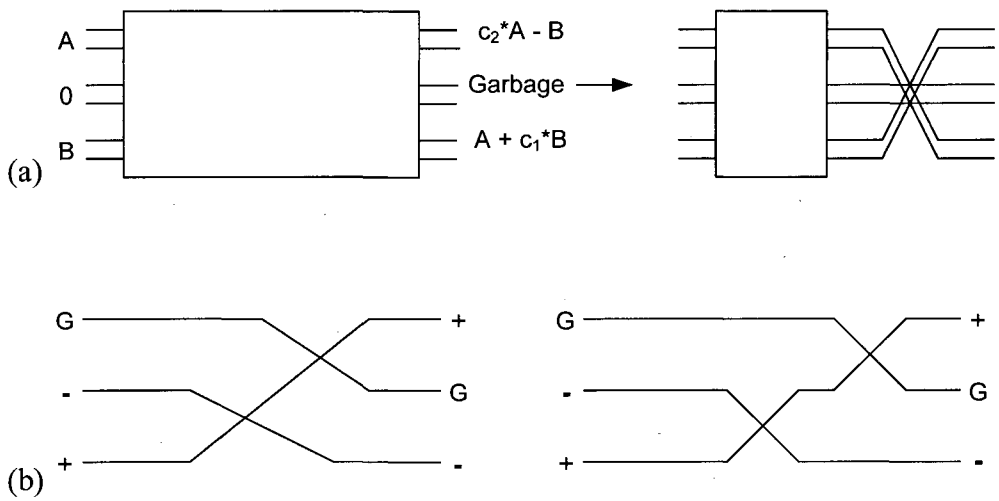
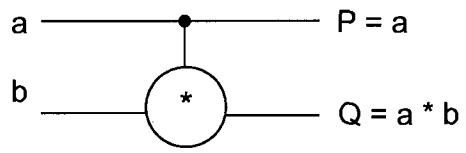


Figure 11.12.2: Realization of the kernel block for the Generalized Transform Butterfly, (a) the location of qubit buses in the diagram, (b) Another variant of SWAPs for the circuit obtained from Figure 11.12.1b by removing SWAPs from the right. The left part shows symbolic SWAPs, the right part the rewritten diagram, allowing to map crossing connections to sequences of SWAP gates, similarly as it was done in Figure 11.11.6.

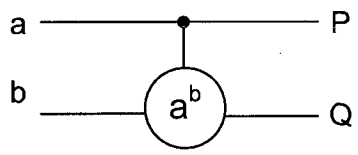


(a)

$$\left.\begin{aligned} P &= a \\ Q &= a * b \end{aligned}\right\} \Rightarrow \begin{aligned} a &= b \\ b &= \frac{Q}{P} \end{aligned}$$

(b)

Figure 11.12.3: Reversible multiplier/divider and the derivation of its equations.

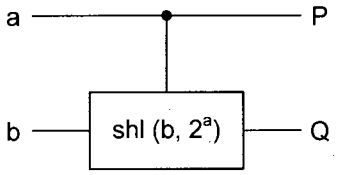


(a)

$$\left.\begin{aligned} P &= a \\ Q &= a^{\,b} \end{aligned}\right\} \Rightarrow \begin{aligned} \log_a a &= b \\ b &= \log_p Q \end{aligned}$$

(b)

Figure 11.12.4: Reversible power/logarithm circuit and the derivation of its equations.



(a)

$$Q = shl(b, 2^a) = 2^a * b \Rightarrow \left.\begin{aligned} P &= a \\ Q &= 2^P * b \end{aligned}\right\} \Rightarrow \begin{aligned} & 2^{-P}.Q = 2^{-P}.2^P.b = b \\ & b = 2^{-P}.Q = shl(Q, 2^{-P}) = shr(Q, 2^P) \end{aligned}$$

(b)

Figure 11.12.5: Reversible shift circuit and derivation of its equations.

Figures 11.12.3 – 11.12.5 present new reversible word-level blocks. They all generalize the principle of CNOT gate, used also in Figure 11.11.1b and Figure 11.11.7.

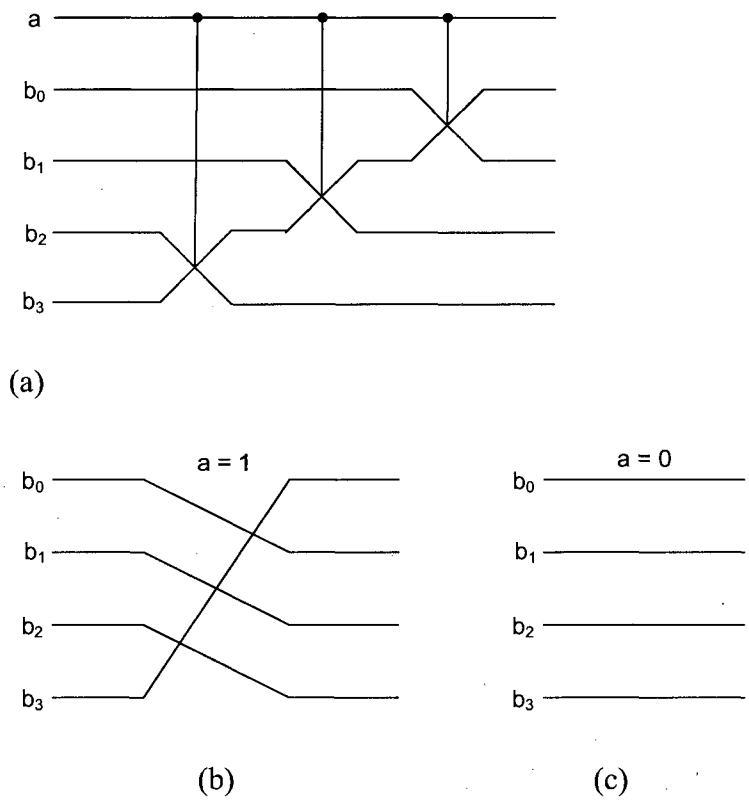


Figure 11.12.6: Cyclic "Shifter To Right" circuit for 4 bits. (a) the quantum array with Fredkin gates, (b) its operation for control qubit a = 1, (c) its operation for control qubit a = 0.

Figure 11.12.6a shows a reversible shifter that shifts to right in forward and shifts to left in backward (output → input) direction or operations. Figure 11.12.6b illustrates its behavior for control a = 1. Figure 11.12.6c illustrates its transparent behavior for control a = 0. Figure 11.12.7 shows right/left cyclic shifter from inputs controlled by two inputs a and b.

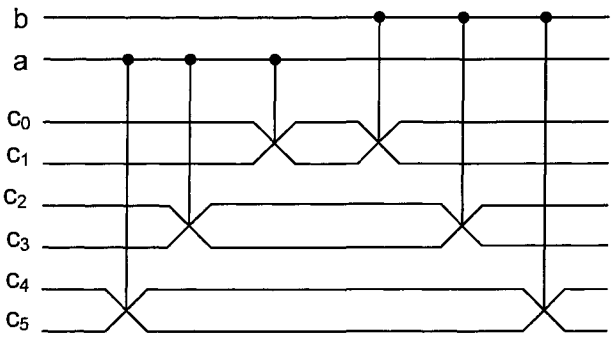


Figure 11.12.7: Left/right reversible cyclic shifter.

The operation of this general shifter is described with the following equations:

a = 0, b = 0 or a = 1, b = 1 – no shift.

a = 1, b = 0 Cyclic shift right.

a = 0, b = 1 Cyclic shift left.

Similarly, all kinds of barrel shifters can be considered to create their reversible (quantum) counterparts.

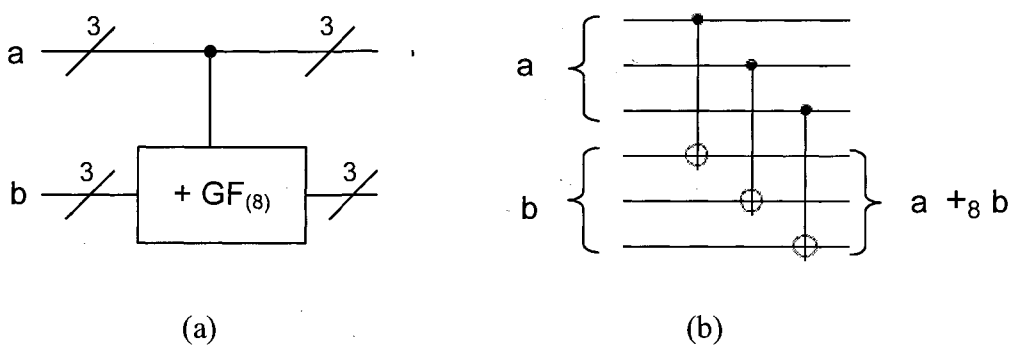


Figure 11.12.8: (a) The schematic of GF(8) adder realized in Binary, (b) The quantum array for GF(8) adder.

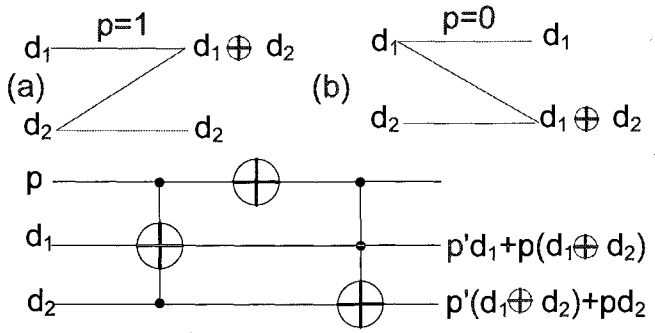
Finally, Figure 11.12.8 presents the simplicity of GF(k) adder for $k = 2^r$ and $r = 3$. Such circuits are used in DSP, communication and cryptography.

11.13. Low Level Realization of FPRM Transforms. FPRM Processor

The butterfly diagrams described in previous chapters for the "fast" calculation of the FPRM spectral coefficients may be represented as quantum logic circuits comprised of cascades of generalized Toffoli gates. Furthermore, all possible butterfly diagrams for any given polarity may be described as a single quantum logic circuit with the polarity number provided as an input to the circuit. Figure 11.13.1 contains the butterfly diagrams for all functions of one variable. The diagram on the left (Figure 11.13.1a) represents the polarity-0 transform while the diagram on the right (Figure 11.13.1b) represents the polarity-1 transform. Values d1 and d2 represent binary truth vectors for all possible functions of 1-variable. The right side of each butterfly expresses the RM spectral coefficients in terms of the original function values. The quantum logic circuit (Figure 11.13.1c) is a realization of the composite function formed using the polarity value p to select which of the two sets of coefficients are requested as shown in the expressions on the right side of the quantum logic circuit.

Just as butterfly diagrams representing RM transforms of more than 1-variable can be formulated based on the Kronecker product, so can the quantum logic circuit also be

expanded for larger functions. Figure 11.13.2 depicts an expanded FPRM processor for 3-variable functions.



(c)

Figure 11.13.1: RM Transformation Butterflies and Corresponding Quantum Logic Circuit. (a) the simple butterfly for polarity 0, (b) the simple butterfly for polarity 1, (c) the circuit for both polarities, polarity is selected by assigning a binary value to variable p.

The FPRM processor accepts a vector corresponding to the Boolean function and a polarity vector and outputs FPRM spectral coefficients. The core part of the FPRM processor is the "butterfly" quantum circuit. The polarity of the "butterfly" is controlled by the polarity bits. Figure 11.13.1 shows the 1-variable FPRM processor which has a 2-bit function input ($[d_1, d_2]$) and a 1-bit polarity input (p). If $p = 0$, the 2-bit output corresponds to positive polarity coefficients, otherwise, if p = 1 the 2-bit output corresponds to the negative polarity coefficients.

Figure 11.13.2 shows the 3-variable FPRM processor. There are 3 polarity bits and 8 input lines for 3-variable processor. Again, I would like to point out that this diagram

is only one example of many possible realizations of various polarity transforms that may be designed based on the "polarity controlled Kernel" concept that was outlined in this section.

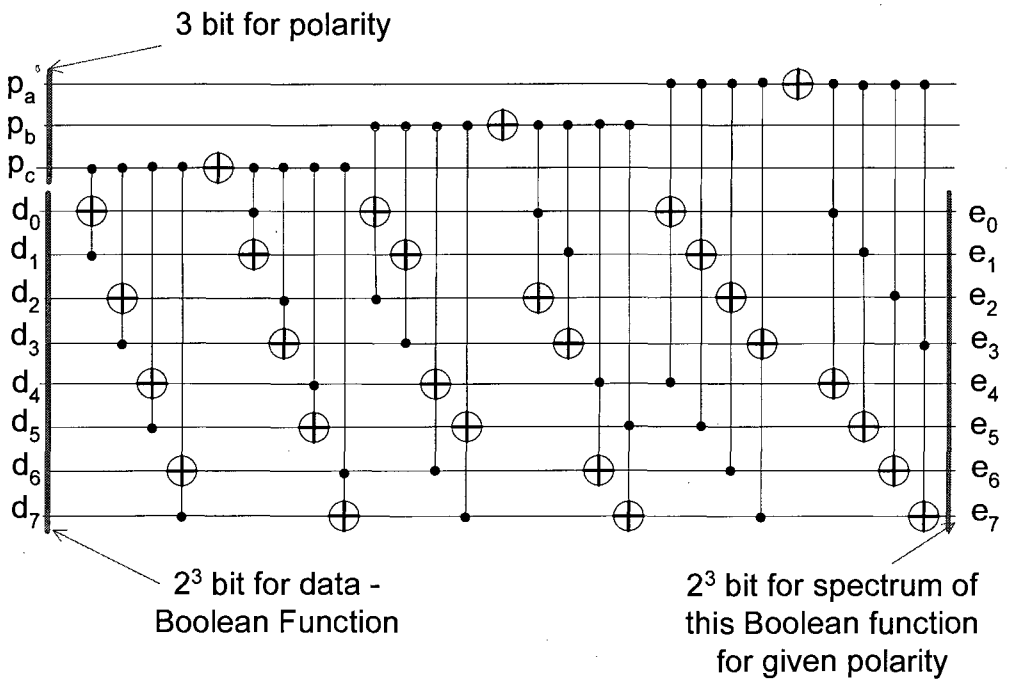


Figure 11.13.2: 3-variable FPRM Processor using butterfly of blocks from Figure 11.13.1.

CHAPTER 12

Quantum Search for Satisfiability, Petrick Function Minimization and Related Problems

In this chapter we discuss how to construct binary and multiple-valued-input oracles for software and hardware realized reconfigurable, tree search, quantum algorithms. We analyzed several algorithms for solving combinatorial problems and we found certain similarities. These similarities were next used to construct general concepts of algorithms based on oracles. These algorithms cover a very wide class of problems. They can be realized in software or in hardware and can model both binary and multiple-valued logic. There is also a didactic value in comparing these algorithms and building oracles for them. We want to create a system for prototyping quantum algorithms and compare them with evolutionary, tree search and other classical algorithms.

In this chapter we show a class of problems that are reduced to a class of oracle-based algorithms: genetic and tree search in case of classical algorithms and Grover algorithm in case of truly quantum algorithm. We hope that our analysis shown in this chapter will allow creating and analyzing various classes of algorithms quickly and with little effort. In contrast to searching an unstructured database application of Grover which is of questionable use, all these applications are practical. They are all

based on creating oracles for Grover algorithm. The first task is then to be able to design oracles systematically and for any problem.

Our first main task is to create quantum oracles for various satisfiability types of problems. This is the topic of chapter 12. We will start from the simplest of these problems and continue towards explanation of a large class of still unsolved problems. Working in a systematic way on designing quantum oracles for many problems, I realized that there are certain categories of problems for which oracles are very similar. Therefore, we tried to categorize all oracles to certain types. Such characterization will allow next to design oracles with less effort, systematically and by reusing reversible blocks for typical circuits.

Based on the solved problems, there exist the following types of oracles:

1. Satisfiability oracles: These oracles are based on creating a single-output satisfiability formula. The formula can use various gate types and logic structures, depending on the problem.

2. Constraint satisfaction oracles: These type of oracles are for constraint satisfaction problems such as graph coloring, image matching or cryptographic puzzles. These oracles use logical, arithmetical and relational blocks and have often the decision oracle and the optimization oracle as their components. The decision oracle is a global AND of several partial decision sub-oracles. The

Constraint Satisfaction Oracles can be treated as generalizations of Satisfiability Oracles. This understanding helped me to build Constraint Satisfaction oracles and I believe it should be always used when new oracles for CS problems are being build.

3. Path problems: These are problems to find certain path in a graph, for instance an Euler path or Hamiltonian path. Many games and puzzles such as "Man, wolf, Goat and Cabbage" belong to this category. The oracles include decision sub-oracles for each move(edge) in the graph of the problem(game). These can be also treated as constraint satisfaction problems in which constraint variables are repeated for units of time. When we know how many time units we need in a sequence of moves to solve the problem, the problem can be reduced to the constraint satisfaction problem.

4. Problems related to spectral transforms: Walsh, Reed-Muller, Haar, Fourier, etc.

5. The mapping problems, including their special class, the subset selection problems. These problems are also constraint satisfaction problems, but they have special properties which makes them easier to solve based on analogies to similar problems.

The Satifiability oracles include the following:

1. POS satisfiability,
2. Solving the unate covering problem by Petrick Function,
3. Solving binate covering problem,
4. Solving various multi-level SAT formulas, especially the generalized SAT of the

   form $\prod\sum\prod_{x_i}$
5. Solving the even-odd covering problem for ESOP, PPRM, FPRM and similar logic minimization problems,
6. Solving the AND-OR Directed Acyclic Graphs (DAGs) from robotics and Artificial intelligence.

The constraint satisfiability oracles include:

1. Proper graph coloring
2. Compatible graph coloring
3. Graph coloring problems with non-standard cost functions
4. Waltz algorithm for image matching
5. Cryptoarithmetic puzzles such as SEND + MORE = MONEY

The Mapping oracles include:

1. Maximum cliques (used in Maghoute algorithm for graph coloring),
2. Maximum independent set,

3. Finding prime implicants of a Boolean Function.

Path oracles include:

1. Euler path,
2. Hamiltonian path,
3. Shortest path,
4. Longest path,
5. Traveling salesman path,
6. Missionaires and cannibals logic puzzle,
7. Man, Wolf, Goat and Cabbage logic puzzle.

Exhaustive solving of equations includes:

1. $a^n + b^n = c^n$

12.1. Solving the Satisfiability Class of Problems

12.1.1. Product of Sums SAT (POS SAT)

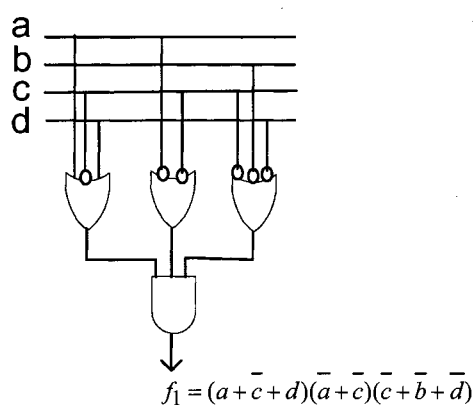


Figure 12.1.1.1: Classical oracle for POS Satisfiability $f_1 = (a + \bar{c} + d)(\bar{a} + \bar{c})(\bar{c} + \bar{b} + \bar{d})$.

The fundamental role of satisfiability to computer science, algorithm design, CAD and complexity theory is well-known. In our ECE-572 and ECE-573 classes at PSU we make many uses of it.

In this section we will present examples of building quantum oracles for various satisfiability problems. Let us build first the oracle for function $f_1 = (a + \bar{c} + d)(\bar{a} + \bar{c})(\bar{c} + \bar{b} + \bar{d})$. The classical oracle is presented in Figure 12.1.1.1. The formula can be transformed as follows, using De Morgan rules:

$$\overline{\overline{(a + \bar{c} + d)} \bullet \overline{(\bar{a} + \bar{c})} \bullet \overline{(\bar{c} + \bar{b} + \bar{d})}}$$
$$= \overline{\overline{acd}} \bullet \overline{ac} \bullet \overline{cbd} \qquad \textit{Equation 12.1.1.1}$$

Using Equation 12.1.1.1 the quantum oracle can be build as shown in Figure 12.1.1.2. Unfortunately this method requires many ancilla qubits and nothing can be done about those qubits if the designer is not performing some deeper transformations of function f_1.

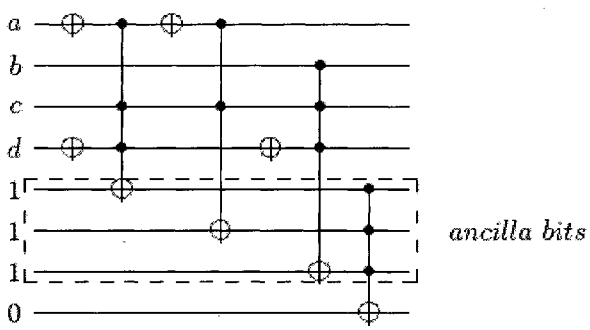


Figure 12.1.1.2: Realization of oracle for POS SAT $f = (a + \bar{c} + d) \bullet (\bar{a} + \bar{c}) \bullet (\bar{c} + \bar{b} + \bar{d})$ *using quantum NANDs and a quantum AND.*

Mirrors to restore ancilla bits to "1" are not shown. These mirrors are not necessary in some applications.

Observe that as many intermediate ancilla bits are required as there are sum terms in the POS formula. This way, every POS SAT formula can be converted to a quantum array with ancilla bits in Figure 12.1.1.2.

12.1.2. Generalized SAT.

In some problems the satisfiability formula is not in POS form. It can be either converted to a POS form, which is often very inefficient, or it can be designed as a multilevel circuit of other structure than that from Figure 12.1.1.2.

For example, given is a SAT formula:

$$f_2 = [(ab + cd) \bullet (ac + \bar{b})] \oplus [(ab\bar{c}d) \bullet (a + b + c)] \qquad \textit{Equation 12.1.1.2}$$

The formula is transformed to the following form

$$f_2 = [(ab \oplus cd \oplus abcd) \bullet (\bar{b} \oplus bac)] \oplus [ab\bar{c}d \bullet (a \oplus \bar{a}b \oplus \overline{\bar{a}}bc)] \qquad \textit{Equation 12.1.1.3}$$

and realized as in Figure 12.1.1.3. As we see, the mirror circuit has been used to decrease the number of ancilla bits. The mirror circuit shown in Figure 12.1.1.3

creates zeros in ancilla bits 2 and 3 from bottom. Another way to realize (a + b + c) would be to use $\overline{\overline{a}\,\overline{b}\,\overline{c}}$. These types of formulas allow for various trade-offs with respect to numbers of ancilla bits.

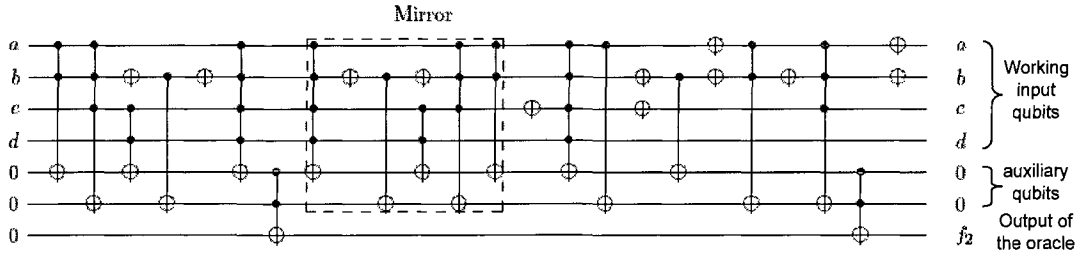


Figure 12.1.1.3: Oracle for function $f_2 = [(ab + cd) \bullet (ac + \overline{b})] \oplus [(ab\overline{cd}) \bullet (a + b + c)]$ *using mirror circuits to decrease the number of ancilla bits. The circuit is not minimized.*

The number of ancilla bits can be reduced by using mirrors when the circuit can be drawn in a layered form of type OR-AND-OR-AND etc. For instance, the circuit from Figure 12.1.1.4a has (from the output) layers of OR, next AND and next OR gates. Using De Morgan rule the circuit is transformed to the form from Figure 12.1.4b and next to the structure from Figure 12.1.1.4c. In this final structure all gates are NANDs. Signals a_i corresponds to ancilla bits. Transforming each gate separately one obtained the (non-optimized) quantum array from Figure 12.1.1.5. As we see, there are 6 ancilla bits (output qubit is mandatory, so it is not counted as an ancilla bit). Mirrors can be used to restore all 6 ancilla bits to constant 1. The question is now the following: Can we decrease the number of ancilla bits? The answer depends on the particular multi-level structure to be realized.

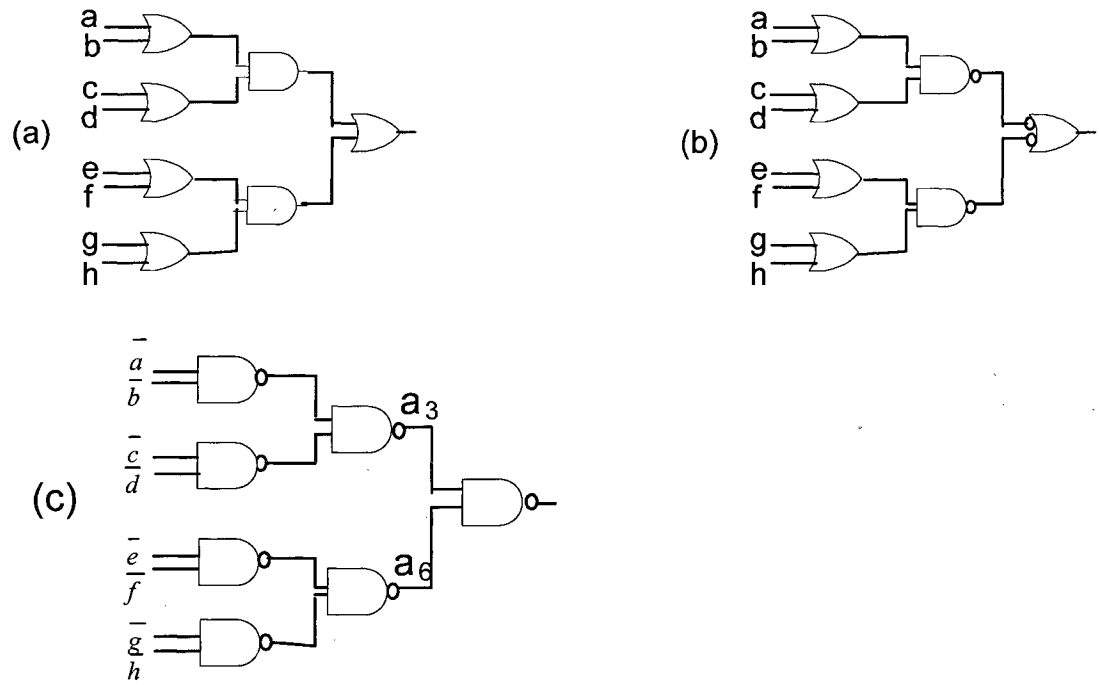


Figure 12.1.1.4: Step-by-step transformations of large classical oracle with many levels to a quantum oracle. (a) Initial oracle with sandwiched layers of OR and AND gates, (b) Converting first K-1 layers to NANDs, (c) converting the last layer to NANDs.

This example illustrates that one can convert an arbitrary formula of Boolean logic to a quantum oracle by adding some number of ancilla bits.

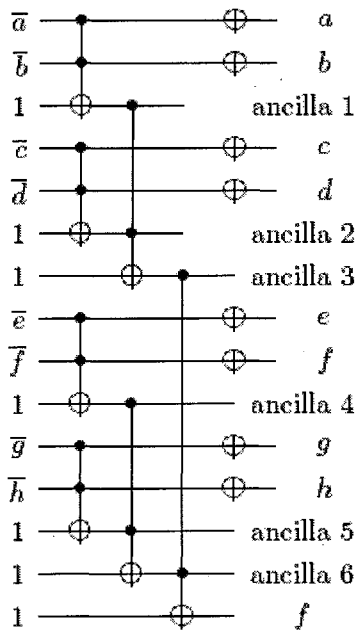


Figure 12.1.1.5: Non optimized quantum array of the classical oracle from Figure 12.1.1.4c. Mirrors can be added to return to constants 1 in all ancilla bits.

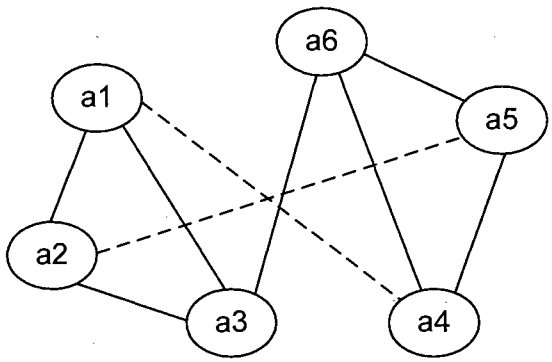


Figure 12.1.1.6: Incompability graph for the ancilla bits from Figure 12.1.1.5. Symbol a1 corresponds to ancilla 1 in Figure 12.1.1.5 and so on. Pairs of Incompatible qubits (those that can not be merged into single qubit) are linked using full edges in the graph. Some pairs of compatible qubits are marked by interrupted lines for illustration.

Figure 12.1.1.6 presents the incompatibity graph for the non-optimized circuit from Figure 12.1.1.5. Every two ancilla bits (nodes) that can not be combined are linked by a solid edge. The graph shows that there are the following maximum independent sets: $\{a_1, a_4\}$, $\{a_1, a_5\}$, $\{a_2, a_5\}$, $\{a_2, a_4\}$, among others. We select pairs $\{a_1, a_4\}$ and $\{a_2, a_5\}$ for folding. Thus ancilla bits a_1 and a_4 are folded to one qubit and ancilla bits a_2 and a_5 are folded to another qubit. This leads to the quantum array with mirror circuit, presented in Figure 12.1.1.7. As we see from this example, the graph coloring, maximum clique and maximum independent set problems are also useful in quantum layout. The maximum clique and maximum clique problems are discussed in details in section 12.3 below.

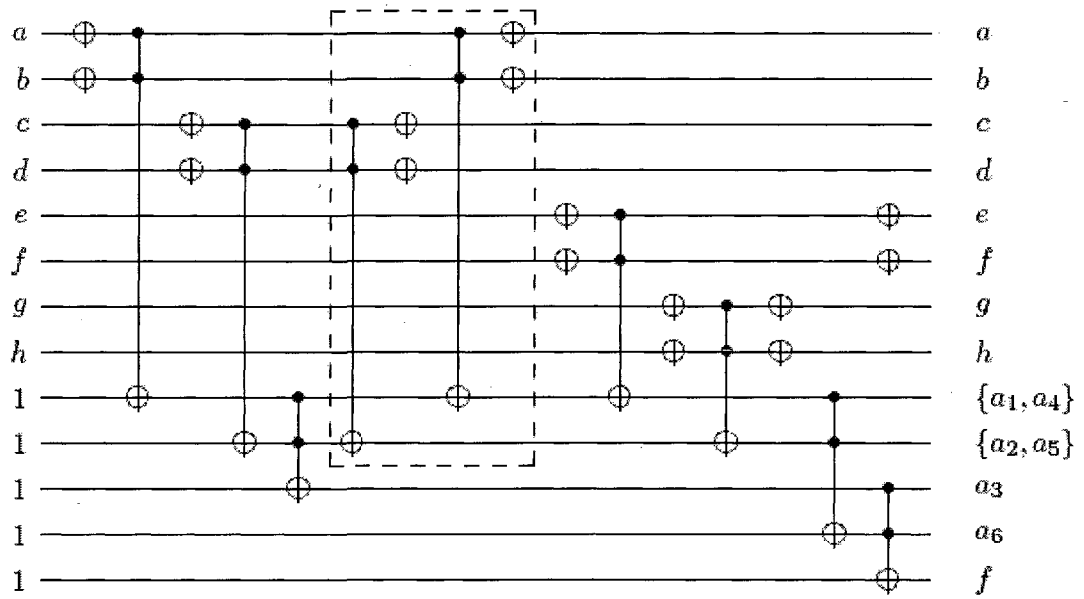


Figure 12.1.1.7: Quantum array for netlist from Figure 12.1.1.4 with mirror a circuit designed based on folding that was found from graph from Figure 12.1.1.6. More mirror circuits can be added to restore bits 2, 3, 4 and 5, counted from the bottom, to constants 1.

12.1.3. AND/OR DAGs.

There are several problems in Artificial intelligence, CAD, planning and scheduling that can be represented by trees or DAGs (directed acyclic graphs). These structures can be converted to satisfiability formulas in classical logic which are next converted to quantum arrays.

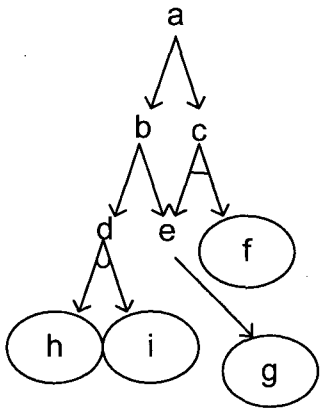
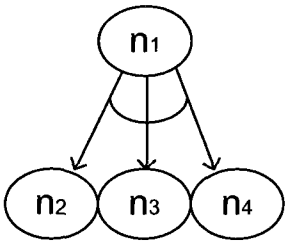


Figure 12.1.3.1: AND/OR DAG for certain Artificial Intelligence Task (such as robot planning). Nodes represent tasks. Leafs represent trivial actions. Arrows represent task dependence. Nodes c and d are AND-nodes. Others are OR-nodes. Node e is implication node and nodes h, i, g and f are terminal nodes (leafs).

Given is for instance a DAG from Figure 12.1.3.1, called the AND-OR graph—the data structure used in AI. There are two types of nodes in Figure 12.1.3.1 – the AND nodes, denoted by an angle symbol between outgoing edges. This means that to satisfy the parent node, all its children nodes must be satisfied, see Figure 12.1.3.2.

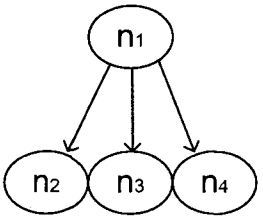


$$\Leftrightarrow (n_1 \rightarrow n_2 \bullet n_3 \bullet n_4)$$
$$(\overline{n_1} + n_2 n_3 n_4)$$

(a) (b)

Figure 12.1.3.2: Example of the AND node in the AND/OR graph. (a) the subgraph, (b) the logical transformation to remove the implication operator.

The other type of nodes are OR nodes. They mean that to satisfy the parent node, any of its children should be satisfied, see Figure 12.1.3.3.



$$\Leftrightarrow (n_1 \rightarrow n_2 + n_3 + n_4)$$
$$(\overline{n_1} + n_2 + n_3 + n_4)$$

(a) (b)

Figure 12.1.3.3: Example of the OR node in the AND/OR graphs. (a) the subgraph, (b) the logical transformation to remove the implication operator.

From the graph from Figure 12.1.3.1 the following logic equation is written:

$$(a \rightarrow b + c)(b \rightarrow d + e)(c \rightarrow e \bullet f)(e \rightarrow g)(d \rightarrow h \bullet i) \qquad \textit{Equation 12.1.3.1}$$

By using the logic transformation rule

$$(A \rightarrow B) \Leftrightarrow (\overline{A} + B) \qquad \textit{Equation 12.1.3.2}$$

Equation 12.1.3.1 is converted to Equation 12.1.3.3 below

$$\bar{a}+b+c)(\bar{b}+d+e)(\bar{c}+e \bullet f)(\bar{e}+g)(\bar{d}+h \bullet i) \qquad \textit{Equation 12.1.3.3}$$

Applying the OR-to-EXOR transformation $(A+B) \Leftrightarrow (A \oplus \bar{A}B)$ the following Equation 12.1.3.4 is created:

$$(\bar{a} \oplus ab \oplus a\bar{b}c) \bullet (\bar{b} \oplus bd \oplus b\bar{d}e)(\bar{c} \oplus cef) \bullet (\bar{e} \oplus eg)(\bar{d} \oplus dhi) \qquad \textit{Equation 12.1.3.4}$$

It is now easy to create an oracle for the function from Equation 12.1.3.4, using in general the methods already outlined in this thesis; including mirrors and factorizations, and possibly, ancilla bits.

12.2. Solving the Unate Covering Problem.

Given is a function from Figure 12.2.1. All its prime implicants are marked as ovals(loops). Using the minterm compatibility graph G all primes are found as maximum cliques. They can be also found as maximum independent sets of graph $\bar{G}$ (G complement). Based on KMap and primes we can create the covering table from Figure 12.2.2.

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

Figure 12.2.1: Finding graphically all prime implicants for minimal Covering of a SOP circuit.

		0001	0101	0111	0110	1100	1101	1111	1011
A	$\bar{a}c\bar{d}$	X	X						
B	$\bar{a}bc$			X	X				
C	acd							X	X
D	$ab\bar{c}$					X	X		
E	bd		X	X			X	X	

Figure 12.2.2: Covering table for function from Figure 12.2.1.

From the table, denoting rows A, B, C, D, E we compile the Petrick function in a standard way:

$$1 = A \cdot (A + E) \cdot (B + E) \cdot B \cdot D(D + E) \cdot (C + E) \cdot C$$

This function can be simplified using the Boolean law as follows :

$1 = A \bullet B \bullet D \bullet C$

Therefore,

$$f = A + B + C + D = \overline{ac}\,\overline{d} + \overline{a}bc + acd + ab\overline{c} + bd$$

Another search method for (another) unate covering table from Figure 12.2.4 is illustrated in Figure 12.2.3. Figure 12.2.3 shows the branching tree for the unate covering problem from Figure 12.2.4. All leafs are solutions, as showed in Figure 12.2.3. Both these methods can be used to build search oracles, as well as hybrid parallel searches.

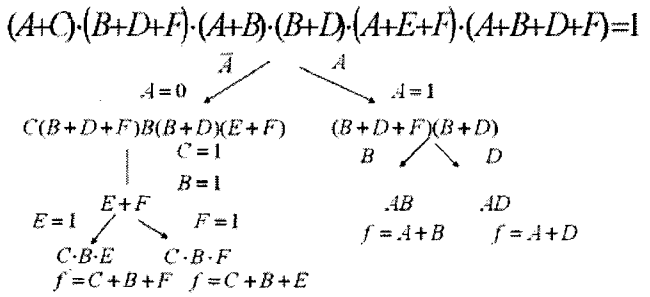


Figure 12.2.3: Solving the Petrick Function from the unate covering table in Figure 1 2.2.4.This method can be combined with oracle methods using the mixed parallel appr oaches from chapter 6.

A	X		X			X
B		X	X	X		X
C	X					
D		X		X		X
E					X	
F		X			X	X

Figure 12.2.4: Another example of an unate covering problem represented by a table.

12.3. Finding Maximum Independent Sets in a graph

12.3.1. The Maximum Independent Set Problem

Finding all Maximum Cliques of a graph and finding all maximum Independent Sets of a graph are two fundamental problems for which creating oracles is relatively easy, so we start from these problems. The complement graph of graph G is a graph with N nodes that when added (set theoretical union of edges) to graph G make a complete clique graph on N nodes.

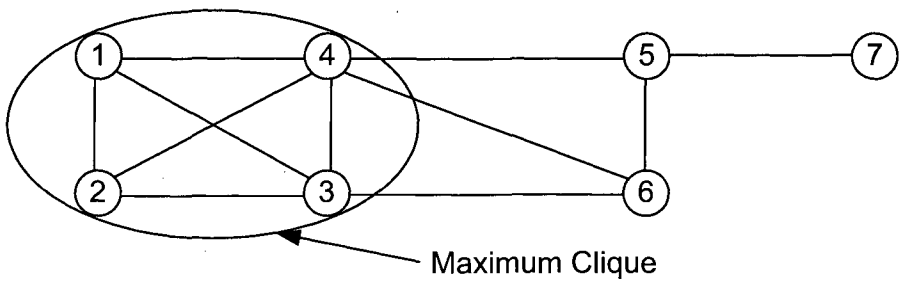


Figure 12.3.1: Maximum Clique in graph G . There are other maximum cliques but this is the only one maximum clique with four nodes. {3, 4, 6} is a maximum clique with 3 nodes.{4, 5, 6} is another maximum clique with 3 nodes. {5, 7} is a maximum clique with 2 nodes.

The standard reversible oracle for finding all independent sets of graph $\overline{G}$ (the complement of the graph G) is given in Figure 12.3.2. It is modified to a quantum oracle for Grover Algorithm in Figure 12.3.3. This method based on mirrors is used always for Grover's Oracle, so the complete quantum oracles with both the base oracle and its mirror will be usually not shown. More details and explanations how to create the oracle from Figure 12.3.2 are given in section 12.3.2.

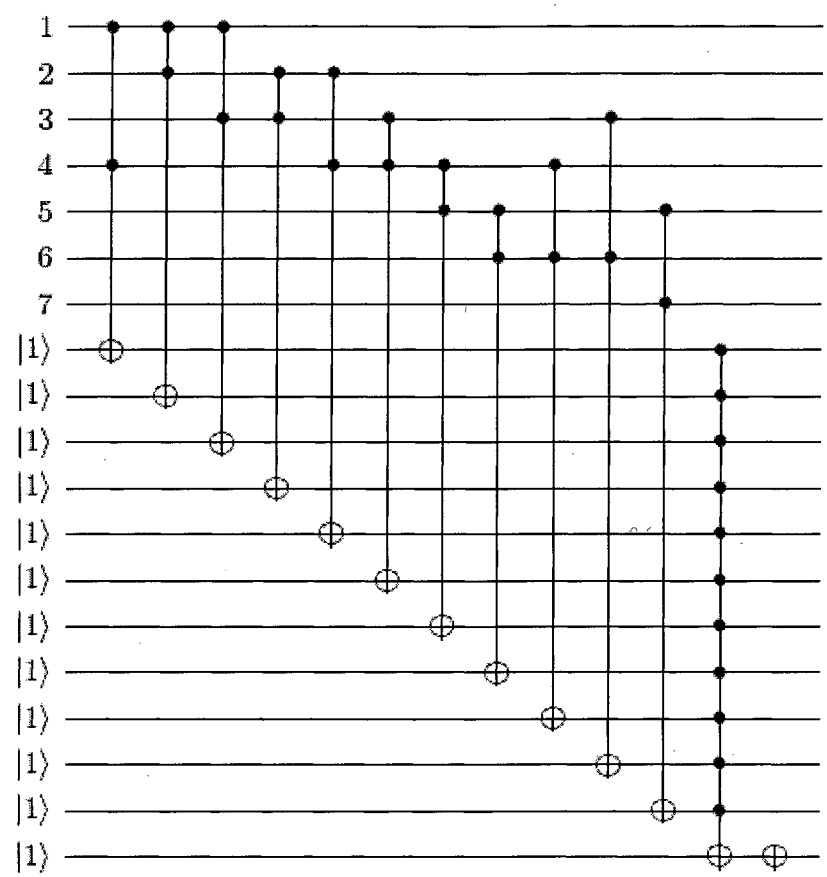


Figure 12.3.2: Quantum Oracle for finding all independent sets of the graph from Figure 12.3.1. All maximum independent sets are found by adding the set size calculating circuit similar to the circuit to calculate the number of colors in a graph from chapter 13.

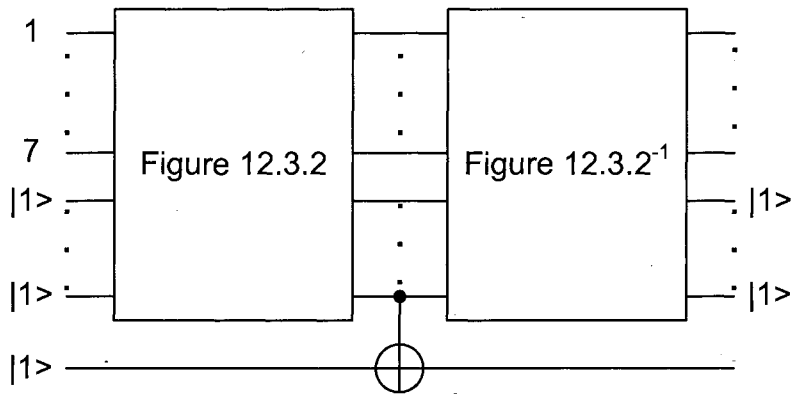
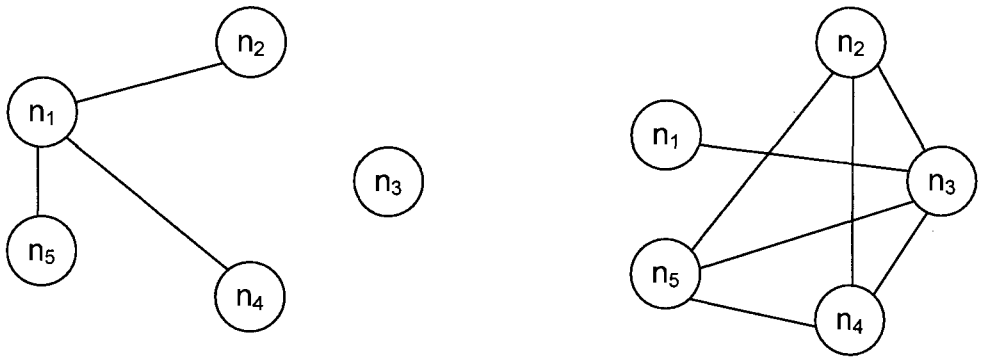


Figure 12.3.3: Using mirror circuit in the oracle for finding all independent sets.

12.3.2. Finding Prime implecants of Boolean Function.

Prime implicants are found from the cliques of the graph G of compatibility of true minterms or from the maximum independent set of its complement graph $\overline{G}$.



Graph G Graph $\overline{G}$

Figure 12.3.2.1: Graph G and its complement graph $\overline{G}$. The maximum independent sets of graph G are the maximum cliques of graph $\overline{G}$, and vice versa. For instance $\{ n_2, n_3, n_4, n_5 \}$ is a maximum clique with four elements.

The Grover algorithm will produce all cliques of size n, next all cliques of size n + 1, etc.

Oracle for the decision part of the maximum independent set for graph from Figure 12.3.2.2 is shown in Figure 12.3.2.3. The optimization Oracle finding all independent sets with more than val nodes is given in Figure 12.3.2.4. This circuit uses the "Count Ones" circuit and "$\leq$ comparator" and is explained in full detail in chapters 11 and 13.

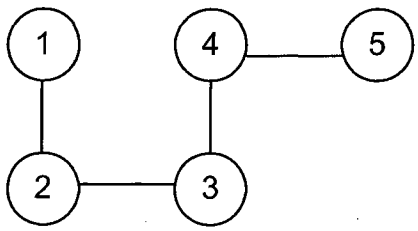


Figure 12.3.2.2: Example of a graph to find the maximum independent set.

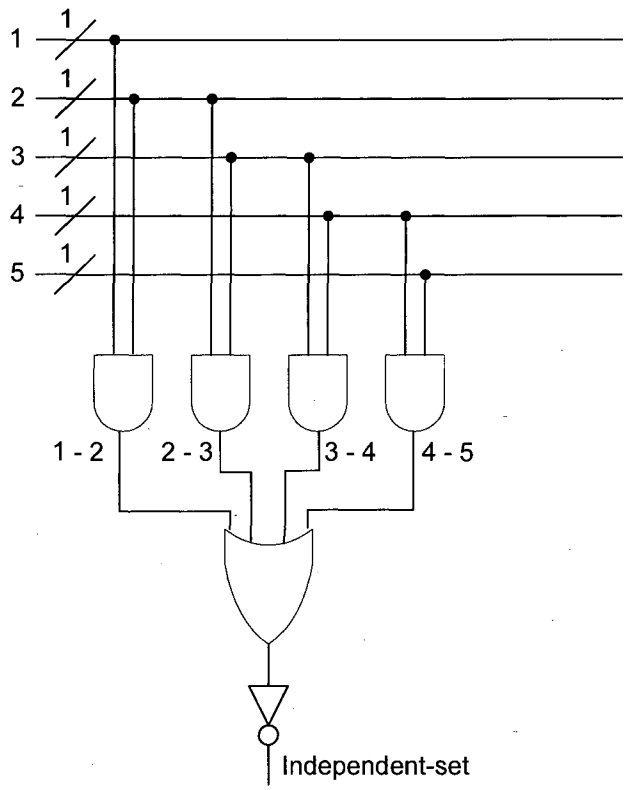


Figure 12.3.2.3: The classical oracle to find all maximum independent sets of graph from Figure 12.3.2.2. Each AND-gate is for an edge of this graph.

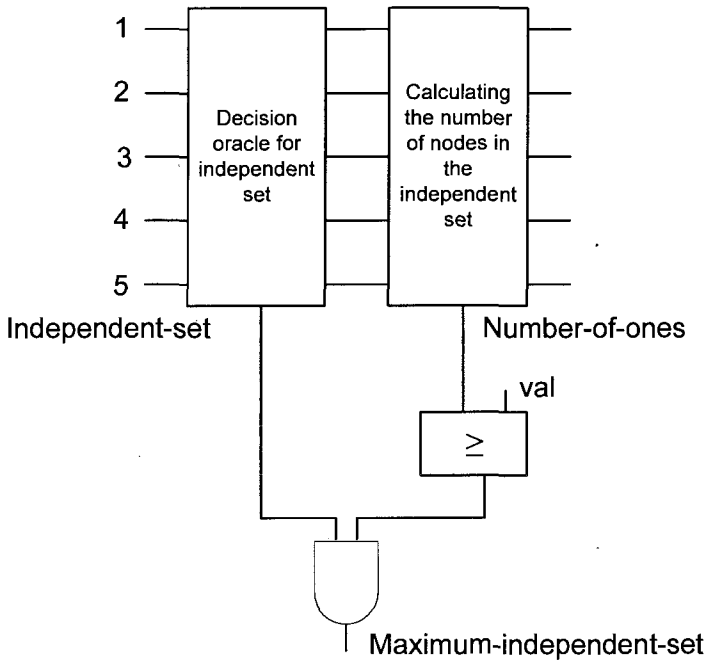


Figure 12.3.2.4: The optimizing Oracle to find all independent sets in a graph that have more than val nodes each.

Another approach to Petrick function minimization is to create an oracle as in Figure 12.3.2.5. One more approach is given in Figure 12.3.2.6. Both these examples use realization of single-index symmetric functions realized as reversible blocks. These way optimization problems such as Petrick Function are converted to decision problems.

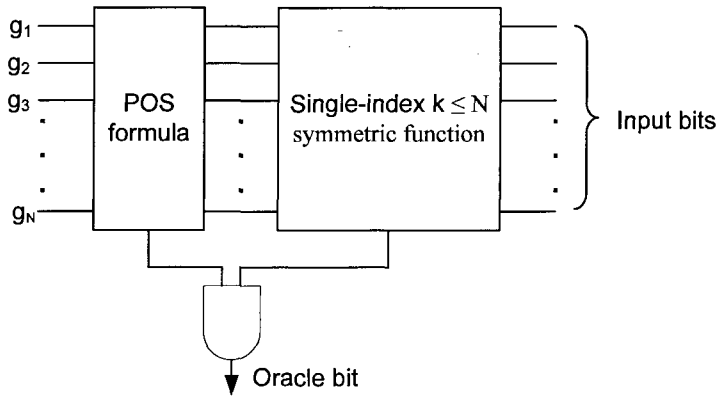


Figure 12.3.2.5: An oracle to solve Petrick Function. Value of k is set by the user.

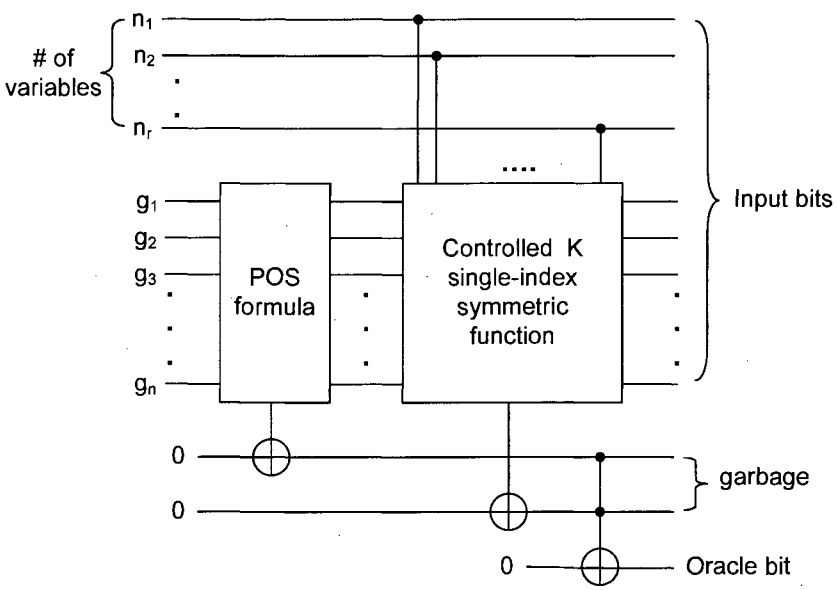


Figure 12.3.2.6: One more alternative approach to solving Petrick Function.

12.4. Classes of Satisfiability Problems.

We showed in sections 12.2 and 12.3 examples of problems that belong to the "satisfiability class of problems". In this section we analyze the satisfiability class of problems in more detail.

12.4.1. Variants of reducing various problems to SAT.

Satisfiability type of problems are the simplest problems for which oracles can be built. Formulating the oracles is quite straightforward from the SAT formula. In the most general case the Satisfiability Decision Function problem is formulated as an arbitrary binary or multiple-valued-input binary-output discrete single-output function, for instance, a product of sums of literals. (The literals are variables negated or not). Another example may be EXOR of products of literals, or product of EXORs of literals, or product of sums of products of literals. These functions are created by transforming some natural-language or mathematical decision problems, such as for instance cryptographic puzzles. The question is to find out for which values of variables the formula is satisfied. In some problems one has to find all solutions, in some other problem just one solution or only some solutions, Often only one solution is enough. An example of oracle for unate function f from section 12.3.2 is shown in Figures 12.3.2.3 and 12.3.2.4. Let us observe that this is a purely logic oracle that gives yes/no answer only. There is a single wire for each variable of the formula. When the function is satisfied the output variable has value "1". The input values may be:

(1) given in exhaustive way by counting,

(2) generated randomly,

(3) generated according to Genetic Algorithm, or

(4) given in a superposition as in quantum algorithms.

Below we will formulate systematically several satisfiability types of problems, starting from the simplest ones.

Given is a product of terms, each term being a Boolean sum of literals, each literals being a Boolean variable or its negation. We are interested in the following problems.

Problem 12.4.1.1 (Satisfiability):

Answer Yes if there exists a product of literals that satisfies all terms or No if such product does not exist.

***Problem 12.4.1.2 (*Optimization of the Generalized Petrick function):**

Find a product with the minimum number of literals that satisfies all terms or (option) prove that such product does not exist.

Problem 12.4.1.3 (Optimization of the Generalized Petrick function-non-negated literal variant):

Find such product of literals that satisfies all terms and in which a minimum number of literals is not negated or prove that no such product exists. (The not negated literals will be also called positive literals).

Problem 12.4.1.4 (Partial Satisfiability):

Find such set of literals that satisfies the maximum number of terms.

Problem 12.4.1.5 (Complementation of Boolean function):

Given is a Boolean function in a Sum of Products Form. Find its complementation in the same form.

Problem 12.4.1.6 (Tautology Checking):

Verify whether a function is a Sum of Product Form is a Boolean tautology.

Problem 12.4.1.7 (Convertion from Sum of Product Form (SOP) to Product of Sums Form (POS)):

Convert a Boolean function from a Sum of Products to the Product of Sums Form.

Problem 12.4.1.8 (Convertion from Product of Sums Form to Sum of Products Form):

Convert a Boolean function from a Product of Sums to the Sum of Products Form.

In problems 12.4.1.2, 12.4.1.3 and 12.4.1.4 we can look for all solutions, all optimal solutions, some optimal solutions or for a single optimal solution. Problem 12.4.1.2 in

which all solutions are looked for corresponds to the well-known Boolean Complementation Problem that occurs in the minimization of Boolean functions.

Other problems such as Tautology Checking, Convertion from Sum of Product Form to Products of Sums Form and Convertion from Product of Sums Form to Sum of Products Form are also mentioned.

The central role of the first problem in Computer Science is well established. Many reductions of practically important problems to problems 12.4.1.2 and 12.4.1.3 were shown, including problems from VLSI Design Automation, especially in logic design and state machine design. It has many applications also in logistics, scheduling, AI and robotics.

Ashenhurt/Curtis Decomposition of Boolean functions can be done in an algorithm that repeatedly applies Satisfiability, Tautology and Complementation. These operations are also of fundamental importance in most algorithms for Boolean minimization, factorization, and multi-level design.

The problem of Partial Satisfiability and its applications are discussed by K. Lieberherr [Lieberherr81, Lieberherr83]. Many other reductions to the formulated above problems are discussed in papers [Garey79, Perkowski 80, Perkowski 86, Perkowski87]. Professor Marek Perkowski created a design automation system [Perkowski85] in which many problems were first reduced to the few selected

"generic" combinatorial optimization problems. These problems include the eight problems introduced above. He was looking to various methods to implement these generic combinatorial algorithms with the goal of finding ones that are as efficient as can be realistically achieved for NP-hard problems with software and hardware technologies of that time. Systolic processors, hardware accelerators and classical oracles were proposed for these problems.

The covering problem is reduced to the minimization of Petrick Function. An example of Petrick function for a covering table from Figure 12.2.4 was shown in Figure 12.2.3. All NP- complete combinational decision problems are equivalent to the Satisfiability Problem [Garey79]. The reductions of many practically important NP-hard combinatorial optimization problems can be also found in the literature. For instance the minimization of the Sum of Products Boolean functions can be reduced to the Covering Problem [Breuer72, Perkowski80] and Covering Problem can be further reduced to the Petrick Function Optimization Problem (PFOP) [Slagle70]. Many other problems, like test minimization can be also reduced to the Covering Problem [Kohavi78, Breuer72, Perkowski80].

The problem of minimization of Finite State Machines includes: (1) the Maximum Clique Problem and (2) the problem of finding minimum closed and complete subgraph of a graph (Closure/Covering Problem) [Perkowski 76]. The first of these problems, (1), can be reduced to the Petrick Function Optimization Problem (PFOP).

The problem of optimum output phase optimization of PLA [Sasao84] can be reduced to PFOP.

The second problem, (2), can be reduced to the Generalized Petrick Function Optimization Problem (GPFOP), introduced above. Many other problems, like AND/OR graph searching [Nilsson71] or TANT network minimization [Gimpel67] were reduced to the Closure/Covering Problem.

A number of problems (Including Boolean Minimization [Perkowski80], [Nguyen87], Layout Compaction [Perkowski80], and minimization of the number of registers in hardware compilation [Perkowski80] can be reduced to the Minimal Graph Coloring Problem. The Minimal Graph Coloring can be reduced to the Problem of Finding the Maximum Independent Sets and next the Covering Problem (Maghoute algorithm). The Problem of Finding the Maximum Independent Sets can be reduced to PFOP. The PFOP is a particular case of the GPFOP. As we then see, all the above problems can be reduced to the Generalized Petrick Function Optimization Problem for which we will create a Grover-based quantum Oracle. A role and importance of Complementation [Sasao85], Tautology [Sasao84b] and Convertions from SOP to POS and vice versa in logic design are well known.

In this chapter we systematically introduce the methods to design Grover oracles and parallel quantum computers (algorithms) for these problems. The conversion of all

these oracles to quantum oracles can be done using methods from this and previous chapters.

12.4.2. Quantum Computers for Solving Satisfiability and Petrick Function Problems

12.4.2.1. A General Characteristics of the Existing Algorithms.

The analysis of various algorithms for satisfiability can be found in the papers of Davis and Putnam [Davis85], Goldberg, Purdom and Brown [Goldberg82], Franco [Franco85], Lieberherr [Lieberherr81, Lieberherr82]. Although it is not generally acknowledged, the Boolean complementation problem [Sasao85, Brayton84] is basically the same as the Problem 12.4.1.2 with all solutions looked for. Various algorithms for solving the Covering Problem and Boolean minimization [Slagle70], [Schmidt74] are basically the algorithms to solve the PFOP, and can be easily adapted to solve the GPFOP.

Most algorithms for these problems from literature known to us are sequential, few are parallel. One is quantum. All the above problems are strongly interrelated. The algorithms to solve them can be basically divided into the following categories:

- tree searching algorithms (ex. Slagle, Schmidt, Purdom, Davis)
- array algorithms (ex. Quine - McCluskey algorithm to solve the covering problem)

- transformational algorithms (ex. Original method to solve the Petrick functions)

The tree search can be differently organized and various tree searching strategies were proposed (chapter 6). We will use the terminology from Nilsson [Nilsson71]. The tree is composed of nodes and arrows. New nodes are created from the nodes by application of operators. Arrows are labeled by operators. In our case operators correspond to literals or sets of literals. Various search strategies are used to expand trees, they use the cost and heuristic functions to select the nodes for expansions and for the ordering of operators. We assume that the order of expanding the tree in the figures is from left to right.

12.4.2.2. Tree-Search Algorithms for Basic Boolean Problems

12.4.2.2.1. A General Characteristics of the Existing Approaches

The non-optimum algorithms for these problems can be divided into the following categories:

1. Greedy algorithms,
2. Random search algorithms,
3. Incomplete tree-search algorithms (they search only a subset of the solution space),
4. Simulated annealing algorithms,
5. Genetic algorithms.

6. Particle Swarm and Bacteria Foraging algorithms,
7. Hardware simulators,
8. SAT-solvers
9. Quantum.

In all these approaches three representations of a General Petrick Function (GPF) are applied, as well as three basic methods of branching. This gives many basic algorithm variants, out of which only few have been investigated in the literature. In this dissertation only few of these categories are illustrated for QSPS, but the careful reader has now enough knowledge to investigate all possible variants and trade-offs, as illustrated in chapter 6.

Let us take the POS form of a GPF:

$F1 = (a + \bar{b} + \bar{c} + d)(\bar{a} + d + e)(\bar{b} + \bar{c} + e)$

The first representation is a list of terms. A term is a Boolean sum of literals. Each term can be also represented as a list. Function **F1** can be then represented as a list:

F1 = ((a (b) (c) d) ((a) d e) ((b) (c) e))

The same representation will be used for a Sum of Products Form:

$F2 = a\ \bar{b}\ \bar{c}\ d + \bar{a}\ d\ e + \bar{b}\ \bar{c}\ e$

The possible methods of tree branching for our problems are the following:

1. a non-balanced binary tree, where each branching is done for a single variable and its negation. Various variables can be selected for branching in different nodes of the same depth of the tree [Purdom83] and additional rules are used for terms being single literals [Davis62].

2. an arbitrary number of successors in each node, branching is done according to the selected term in this node, all literals from this term lead to some successor nodes [Breuer72], [Perkowski 80].

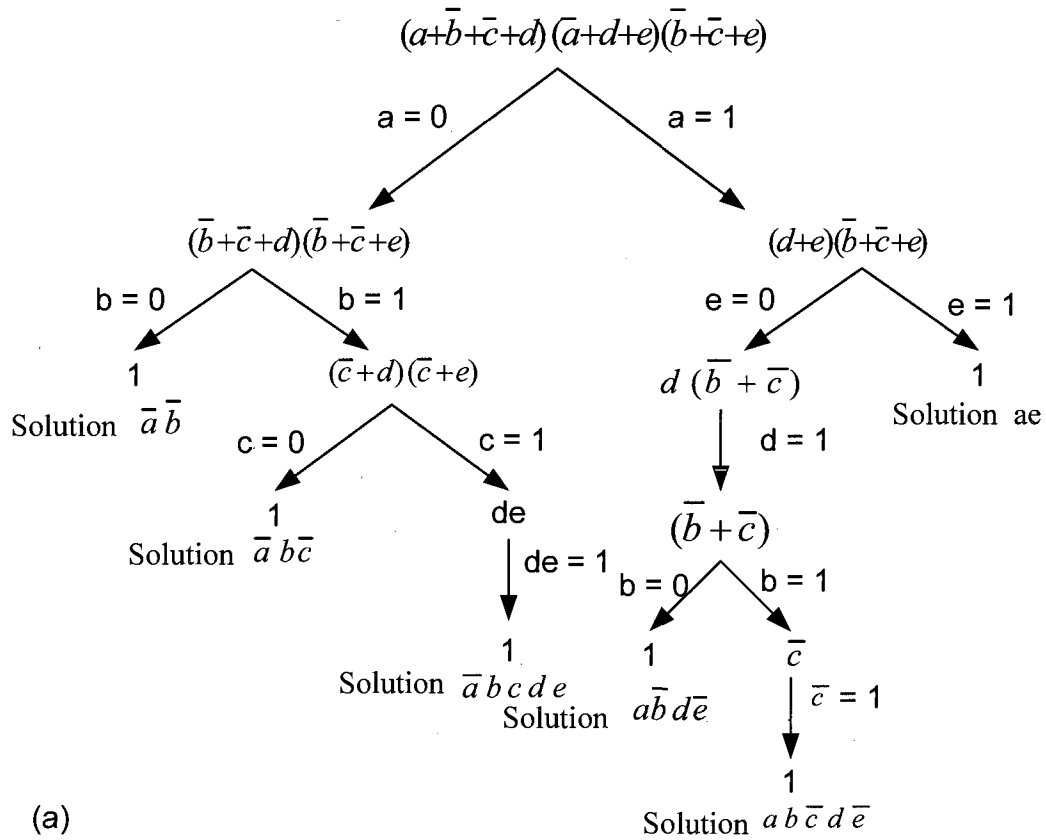
3. a method based on a standard tree of subsets of a set of all literals used in the function [Perkowski80]. This method modifies the standard tree by removing literals that are not present in each current node of the tree.

The method can lead to different problem decompositions of a large SAT problem to many smaller SAT problems with respect to sets of support variables. This can have application in parallel SAT solvers or in quantum SAT solver accelerators with a limited width of the quantum register.

Let us now concentrate on the first branching method only. The following decisions affect speed and quality of solutions obtained from this method.

1. How to select the branching variable?
2. What other rules (like Davis-Putnam) can be applied for creating the operators?
3. How to order the branches of the tree?
4. How to terminate the branches of the tree?
5. What parts of tree are expanded in series and which in parallel?

The modification of the first branching method is shown in Figure 12.4.2.2.1.1. Nodes of the tree correspond to the function and simplified functions that are created after substitutions. The leafs of the tree are the solutions. Usually they correspond to products of operators along the branches. This method is used when all solutions are searched for.



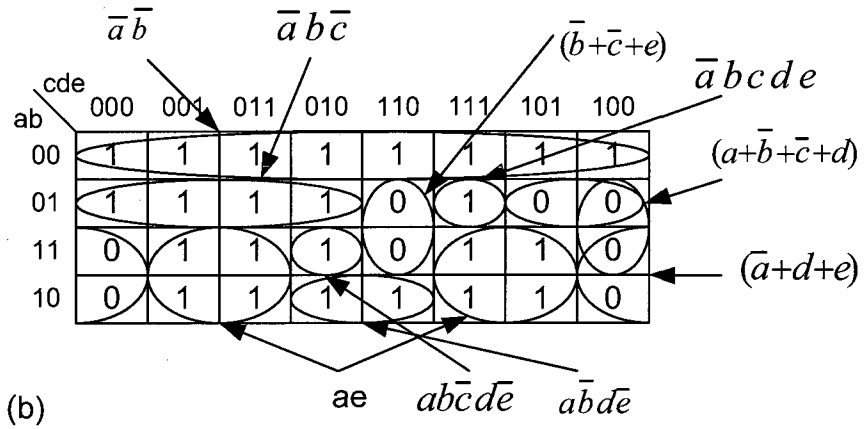


Figure 12.4.2.2.1.1: The variant of the first branching method as the general approach to find all solutions to a SAT, (a) the tree with variable branching and solutions being product groups, (b) the KMap with Sum groups and product groups shown.

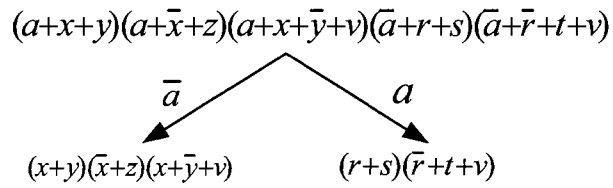


Figure 12.4.2.2.1.2: Smart selection of a decomposition variable in the first branching method.

A variable is selected for branching according to some rule. For instance, a variable can be selected that occurs most often (in both affirmative and negative forms) in the POS formulas of the function. The branching with operators variable = 0 and variable = 1 is done by substituting in the formula the values 0 and 1 for this variable, respectively. Two new nodes are created, in which the corresponding functions are simplified by removing terms with selected literal and removing the negations of the selected literal from other terms. Whenever a term being a single literal is created, it is immediately used for substitution, as in the Davis-Putnam procedure and the algorithms based on it. Figure 12.4.2.2.1.2 illustrates the application of the branching decomposition. The initial function has the support set of 9 variables. After branching for variable a we create two smaller SAT problems, the first with the support set of 4 variables, and the second with the another support set of 4 variables. This way a smaller quantum computer can be build to solve in an exact way each sub-problem in a parallel quantum computer from chapter 6.

From chapters 5 and 6 it should be now perfectly clear that this discussion is general and applies to all kinds of search software, oracles, quantum oracles, and parallel quantum systems such as QSPS.

12.4.2.2. Selection of a Branching Variable

The rules for selecting a variable are:

1. Select a variable that occurs in most terms, in both positive and negative forms.
2. If there are more than one such variables selected in step 1 then select among them variables that occur in the shortest term. If there is only one variable selected in step 2 then return it.
3. If there are more than one variable selected in step 2 then select among them a variable **v** that maximizes the value of the function:

$$CV(v) = \frac{number\ of\ terms\ in\ which\ \text{var}iable\ v\ occurs}{total\ number\ of\ literals\ in\ these\ terms}$$

4. Otherwise return random variable from step 3.

These rules to be used in master processors for hybrid quantum search.

12.4.2.2.3. Additional Operator Selecting Rules

All literals from terms consisting of single literals are selected and no branching is done.

12.4.2.2.3.1. Ordering of Branches

For each variable v selected according to section 12.4.2 we can apply one of two operators: v and $\bar{v}$ as the first operator in branching. The literal that occurs more often in the terms is applied. This leads to solutions being generated earlier when the depth-first search strategy with successors ordering is used, in which the successors of a node are ordered according to the above rule.

These rules are good for sequential parts of parallel algorithms, those that produce initial decompositions of SAT formulas on top of trees (see chapter 6).

12.4.2.2.3.2 Termination of Tree Branches

12.4.2.2.3.3. First variant of branches terminating.

Two new additional rules are used:

1. When a function in a node consists of a single term, the solutions are created for all literals from this term. No branching is done and the current branch is terminated.
2. When all terms in a function include the same literals L1,....Ln and only single other literal each, then the solutions are created for L1,...,Ln and the product of the remaining literals from the terms.
3. No branching is done and the current branch is terminated. When all terms in a function include the same literals L1 ,...., Ln and one or more from these terms include only those literals and no other literals, then the solutions are

created for L1,....,Ln. No branching is done and the current branch is terminated (this is unlike in the well-known procedures).

12.4.2.2.3.4. Second variant of branches termination.

In the optimization problems when only a single optimum solution or some optimum solutions are looked for with a minimum number of positive literals a speed-up can be obtained by using the rules:

[1A.] When a function in a node consists of a single term any literal from this term is selected to the solution. No branching is done and the current branch is terminated.

[2A.] When all terms in a function include the same literals L1, ..., Ln and only single other literal each then the solution is created for L1 . No branching is done and the current branch is terminated. When all terms of the function include the same literals L1,..., Ln and one or more from these terms include only those literals and no other literals then the solution is created for L1. No branching is done and the current branch is terminated.

Additionally, in this type of optimization search problems the cut-off rules are used to backtrack in master processors of parallel systems when the costs of partial solutions

are equal or higher than the current minimum cost (cost of the best solution found until now).

12.4.2.2.3.4. Discussion

The created solutions (so-called implicants of F1) are all different. Comparison of the created products with the Karnaugh map of F1 from Figure 12.4.2.2.1.1b allows us to observe the following properties of this branching method:

1. the complemented function has products which are not neccessarily prime implicants,
2. the implicants in the complemented function are overlapping (are not disjoint)
3. the number of implicants is smaller than the number of all prime implicants of the function (all prime implicants are generated in many methods).

It results from the above that this branching method is well suited for complementation of Boolean functions and for finding of single solutions to the optimization problems. This method is not able to find all optimal solutions to such problems, however, it can produce a subset of quasi-optimal solutions, which in practice can be quite sufficient.

The advantage of this method is that some good solutions can be found with very limited search (using for instance the depth-first tree-search strategy [Nilsson71], and the cut-off in the tree can be applied soon. Another advantage is that each solution is generated only once in the search process.

The main disadvantage of this method is that it may not produce the optimal solution to Problem 12.4.1.2, when the variables are selected in a wrong order. Although in the investigated by us practical examples the solutions were always optimal, they depended on heuristics. It does however provide optimum solution to Problem 12.4.1.3, which has more practical applications.

The solution to Problem 12.4.1.2 is with this branching variant not necessarily optimum since not all combinations of literals are created as branches of the tree. The solution to Problem 12.4.1.3 is optimal since all combinations of positive literals are generated as branches. The other branching methods are compared in [Perkowski87]. Only the implementations of the above two variants of the first method will be discussed below.

12.4.2.2.3.5. Reductions

In this section we will show how some of the problems investigated by us can be reduced to other problems, in order to decrease the number of necessary generic programs in our library of useful CAD routines.

12.4.2.2.3.6. Reduction 1.

Let us assume that we dispose an algorithm **Find_Solutions (Product_of_Sums, Number_of_Solutions, Type_of_Solutions)** that finds solutions (the solutions are products of literals) to **Product_of_Sums**. **Product_of_Sums** is a Boolean function in product of sums form (GPFOP). **Number_of_Solutions** and **Type_of_Solutions** are some user-specified parameters.

When

1. **Number_of_Solutions** = **all**, then all solutions are generated
2. **Number_of_Solutions** = **all_optimal**, then all optimal solutions are generated.
3. **Number_of_Solutions** = **some_optimal**, then some optimal solutions are generated.
4. **Number_of_Solutions** = **one_optimal**, then one optimal solution is generated.

When

1. **Type_of_Solutions** = **literals**, the solutions to minimize the number of literals are looked for
2. **Type_of_Solutions** = **positive_literals**, the solutions to minimize the number of positive literals are looked for.

When this parameter equals nil (empty) the type of the function is irrelevant.

To find a complementation of a Boolean function in a Sum of Products Form it is sufficient to find all solutions to a dual function.

With respect to the representation of Boolean functions shown above the finding of the dual function is trivial. It consists only in negating of all literals in the Sum of Products Form.

F = ab + cd + $\overline{a}\,\overline{b}$

$$\begin{aligned}\overline{f} &= \overline{ab+cd+\overline{a}\,\overline{b}} = \overline{(ab)}\ \overline{(cd)}\ \overline{(\overline{a}\,\overline{b})} \\ &= (\overline{a}+\overline{b})(\overline{c}+\overline{d})(a+b) \\ &= (\overline{a}+\overline{b})(\overline{c}\,a+\overline{c}\,b+\overline{d}\,a+\overline{d}\,b) \\ &= (\overline{a}\,\overline{c}\,b+\overline{a}\,\overline{d}\,b+\overline{b}\,\overline{c}\,a+\overline{b}\,\overline{d}\,a\end{aligned}$$

Equation 12.4.1

Function f in SOP form is represented as ((a b) (c d) ((a) (b))).

(Dual f) in POS form is:

(((a) (b)) ((c)(d)) (a b)) *Equation 12.4.2*

When we generate all solutions for Equation 12.4.2 using the first branch terminating variant and next the set of solutions generated by the program is treated as a SOP then the returned by it complement function will be the same as in Equation 12.4.1.

12.4.2.2.3.7. Reduction 2.

To convert a Sum of Products to Product of Sums Form it is sufficient to find all solutions to this sum of products treated as a product of sums. The result is treated as a sum of products.

Example 12.4.2.2.3.7.1:

Let us take the function from the previous example: f = ab + cd + $\bar{a}\ \bar{b}$. Applying de Morgan Theorem to the right side of the formula Equation 12.4.1 we get:

$$f = \overline{(a\bar{b}\bar{c})}\ \overline{(a\bar{b}\bar{d})}\ \overline{(\bar{a}b\bar{c})}\ \overline{(\bar{a}b\bar{d})} = (\bar{a}+b+c)(\bar{a}+b+d)(a+\bar{b}+d) \qquad \textit{Equation 12.4.3}$$

This is a POS of function f. We treat the SOP of f = ((a b) (c d) ((a) (b))) as a POS and find solutions with the first branch terminating variant. Later we treat the set of solutions as the product of sums. The same solution as in Equation 12.4.3 is found.

Let us verify this. POS corresponding to SOP is:

$$(a + b)\ (c + d)\ (\bar{a} + \bar{b}) = \bar{a}\ b\,c + \bar{a}\ b\,d + a\ \bar{b}\ c + a\ \bar{b}\,d \qquad \textit{Equation 12.4.4}$$

The result Equation 12.4.4 must be treated as POS, then we have:

$$(\bar{a} + b + c)\ (\bar{a} + b + d)\ (a + \bar{b} + c)\ (a + \bar{b} + d) \qquad \textit{Equation 12.4.5}$$

This is the same result as in Equation 12.4.3 .

12.4.2.2.3.8. Reduction 3

Let us assume now that we dispose an algorithm *Satisfiability (Product_of_Sums)* that answers *YES* or *NO* , depending if there is a solution to a *Product_of_Sums*. Let us assume now that we want to check whether some SOP is a tautology. SOP is a tautology when its complement is zero or in other words when the answer to the Satisfiability Problem for the complement is *NO*.

Example 12.4.2.2.3.8.1:

SPF of function f is:

$$F = ab + a\bar{b} + \bar{a}\,b + \bar{a}\,\bar{b} \qquad \textit{Equation 12.4.6}$$

f is represented as ((a b) (a (b)) ((a) b) ((a) (b))) . It can be easily checked that Equation 12.4.6 is a tautology:

$$\begin{aligned} \bar{f} &= \overline{ab + a\bar{b} + \bar{a}\,b + \bar{a}\,\bar{b}} = \overline{(ab)}\ \overline{(a\bar{b})}\ \overline{(\bar{a}\,b)}\ \overline{(\bar{a}\,\bar{b})} \\ &= (\bar{a}+\bar{b})(\bar{a}+\bar{d})(a+\bar{b})\,(a+b) \end{aligned} \qquad \textit{Equation 12.4.7}$$

The right side of Equation 12.4.7 can be calculated by

(Dual Sum_of_Products) = (((a) (b)) ((a) b) (a (b)) (ab))

This is given as an argument to program Satisfiability. Now searching the tree shows that there is no product of literals that satisfies this function. The answer for function from Equation 12.4.7 produced by the Satisfiability program will be *NO*. Therefore the answer to the corresponding Tautology problem will be *YES*.

In conclusion, we will need only three programs to solve all eight problems:

- Satisfiability(Product_of_Sums),
- Find_Solutions(Product_of_Sums, Number_of_Solutions, Type_of_Solution,
- Partial_Satisfiability(Product_of_Sums, Number_of_Solutions, Type_of_Solutions)

These SAT programs can be of any type, including the single oracle system (quantum or classical), and the parallel quantum system.

12.4.2.2.3.9. Other data structures

Let us take the GPF (POS):

F1 = (a + $\bar{b}$ + $\bar{c}$ + d) ($\bar{a}$ + d + e) ($\bar{b}$ + $\bar{c}$ + e)

The first representation is a list of terms. A term is a Boolean sum of literals. Each term can be also represented as a list. Function F1 can be then represented as a list:

F1 = ((a (b) (c) d) ((a) d e) ((b) (c) e))

Its variants use computer words or Boolean cubes [Ulug 74] to represent terms.

The second representation uses an array of symbols 0 and 1 to describe the GPF F1, Figure 12.4.2.2.3.9.1:

a	1	0	0
$\overline{a}$	0	1	0
b	0	0	0
$\overline{b}$	1	0	1
c	0	0	0
$\overline{c}$	1	0	1
d	1	1	0
$\overline{d}$	0	0	0
e	0	1	1
$\overline{e}$	0	0	0

Figure 12.4.2.2.3.9.1: Tabular Representation of Function F1.

This representation uses often arrays of binary words to store rows or columns of the array. The variant of this representation uses half the number rows, but more symbols to be stored in an array are now required. Two bits per symbol (0 , 1 , X , auxiliary E) are used.

	C1	C2	C3
a	1	0	X
b	0	X	1
c	0	X	0
d	1	1	X
e	X	1	1

F1

Figure 12.4.2.2.3.9.2: Second Method for Tabular Representation of Function F1.

The third representation uses lists corresponding to rows of the above arrays:

La = {C1},

$L\bar{a}$ = {C2},

Lb = {C3},

$L\bar{b}$ = {C1},

Lc = {},

$L\bar{c}$ = {C1, C3},

Ld = {C1, C2},

$L\bar{d}$ = {},

Le = {C2, C3},

$L\bar{e}$ = {},

The possible methods of tree branching are the following:

- irregular binary tree, where branching is done for a variable and its negation. Various variables can be selected in different nodes of the tree on the same search depth [Purdom, Haralick],
- arbitrary number of successors in each node, branching is done according to the selected term in this node ,
- all literals from this term lead to successors [Breuer72, Perkowski80],
- standard tree of subsets of a set of all literals used in the function [Perkowski87].

This method modifies the standard tree by removing literals that are not present in each current node of the tree.

12.4.3. Discussion on branching and parallelism.

The first branching method is shown in Figure 12.4.2.2.1.1a. Whenever terms of single literals are created, they are immediately used for substitution, as in the Davis-Putnam procedure and all its successors. The created solutions (implicants of F1) are all different. Comparing the created products with the Karnaugh map of the F1 function (Figure 12.4.2.2.1.1b) permits us to note two properties of this branching method:

- the complemented function has products which are not prime implicants,
- the implicants in the complemented function *are not overlaping*.

It results from the above that this branching method is well suited for complementation of Boolean function and for finding of single solutions to problems. It is not able to find all optimal solutions, however it can produce a subset of quasi-optimal solutions, which in practice can be quite sufficient.

The advantage of this method is that some good solutions are found with small search (using for instance the depth-first tree-search strategy (Nilsson, [Nilsson71]) and the cut-off in the tree can be applied soon. Another advantage is that each solution is generated only once.

The main disadvantage of this method is that it can produce not the optimal solution, when variables are selected in wrong order. Although in the investigated by us practical examples the solutions were always optimal, they depended on the heuristic. The optimum solution {a} is found (Figure 12.4.2.2.1.1) when variable a is selected in the first level (it is selected because variable a occurs most often). When variable c is selected on the first level the best solution found has two literals and is not optimum.

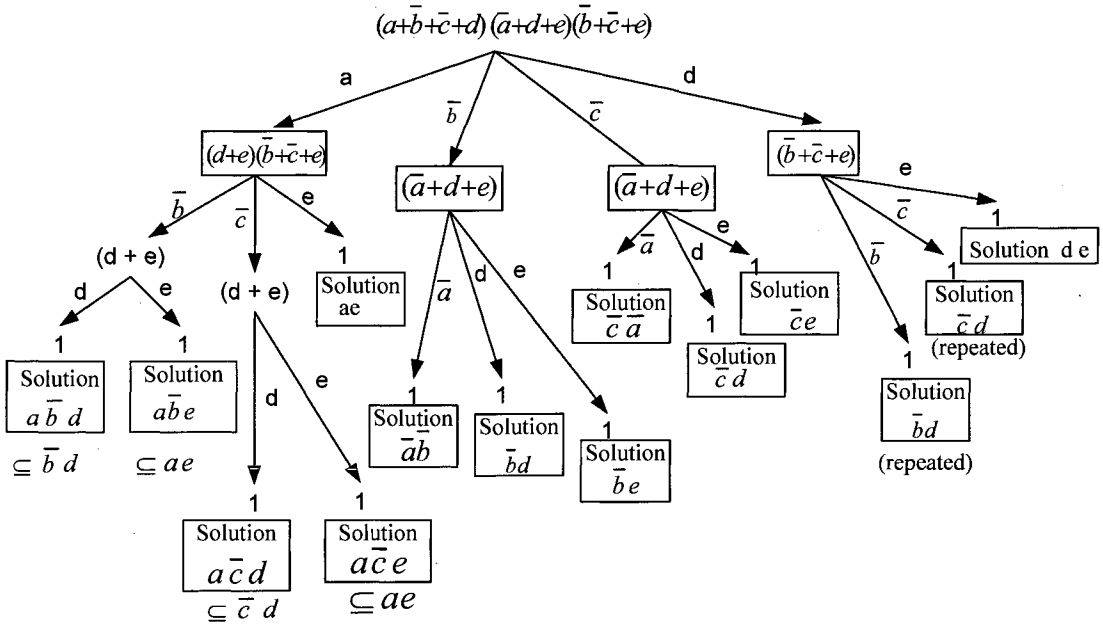


Figure 12.4.3.1: The second branching method applied to function from Figure 12.4.2.2.1.1. Observe that the groups $a\bar{b}d$, $a\bar{b}e$, $a\bar{c}d$ and $a\bar{c}e$ are included in other solutions thus they can be never generated if the search would be executed in another order (like from right to left) and any branch with a group included in the existing group being a solution would be cutted-off.

The *second branching method* is presented in Figure 12.4.3.1. At each node of the tree one term is selected

- according to some heuristic,
- randomly,
- as the first one.

The literals from this term are taken for branching. This method can incorporate not only Davis-Putnam heuristics but also many methods used to solve the covering problem, like dominance of rows or symmetry. As we see in the corresponding Karnaugh map (Figure 12.4.3.2) the created products are prime implicants of the function and they also overlap. The method is then good to generate all prime

implicants while complementing a Boolean function and to generate all optimal solutions to a Boolean function. The disadvantage of this method is that some solutions are generated many times (like in our example). The advantage of this method is that it can generate all optimum solutions.

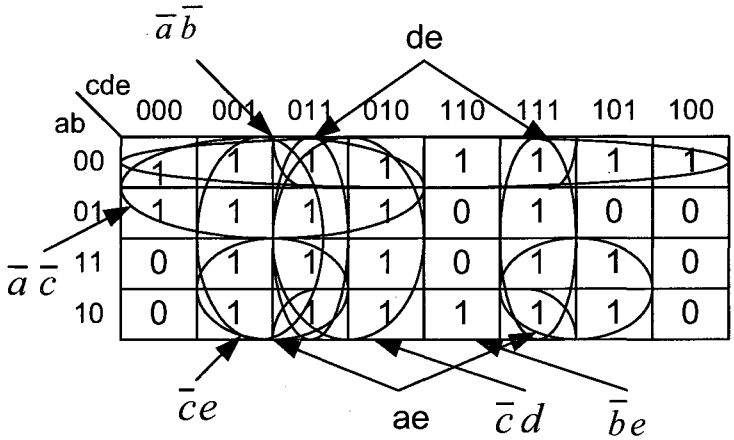


Figure 12.4.3.2: The groups obtained from search in Figure 12.4.3.1 that are not included in other groups generated at the left in Figure 12.4.3.1.

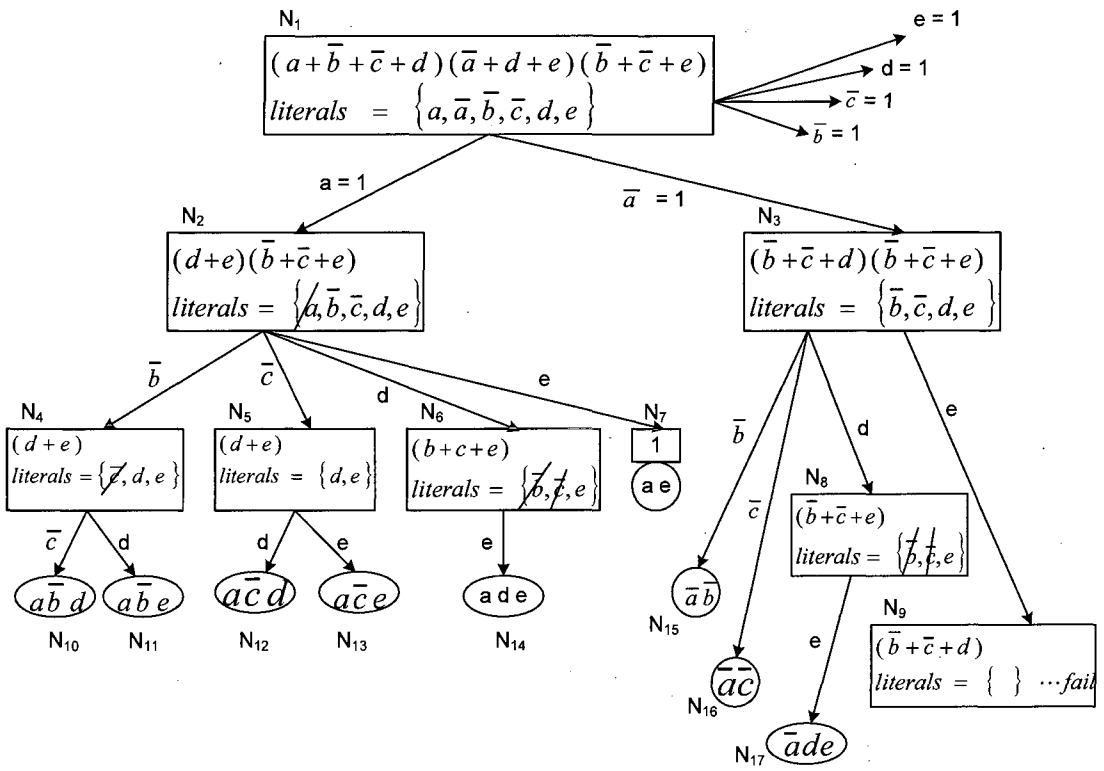


Figure 12.4.3.3: Part of the tree of all subsets of literals applied to function from Figure 12.4.3.1. Observe that solutions included in other solutions are generated and there are solution repetitions. Thus finding new solutions can be speed-up by changing the order of node expansions. Davis-Putnam and other rules can be used to select good expansion nodes. This is shown in nodes N11 and N12.

The third branching method uses the standard method of generating subsets of a set of all literals (see Figure 12.4.3.3). Some literals, like a in node N_1 are cancelled because of search model. However implicants included properly or not into other implicants are generated, which makes this method applicable only if the implicants with costs higher than the cost of the actually minimum solution are cutted-off. This method can generate quickly some product solutions with the minimum number of literals. This method cannot be used to generate all optimum solutions or to complement a Boolean function.

Observe that because of wide branching, the breadth first strategy generates many simple nodes in the first level branching, thus solutions in nodes N_{10}, N_{12}, N_{13} and N_{14} are generated that may be cut-off by earlier finding of node N_7. Similarly node N_{14} can be removed as included in N_7 (ade $\subset$ ae) when we look for all solutions. The tree from Figure 12.4.3.3 is an excellent illustration of various search trade-offs typical for SAT, unite covering, binate covering, even-odd covering, graph-coloring, some mapping and constraint satisfaction problems.

The approximate methods expand some subset of the described above trees. The greedy algorithms find one depth-first path in the tree and if necessary, iterate. The random search algorithms find single random depth-first path and (sometimes) iterate.

The incomplete search algorithms search with some heuristic strategy, that searches only a subspace of the entire solution space. The search strategies include:

- depth-first with limited number of backtracks,
- ordered-search with not-admissible quality function (Nilsson [Nilsson71]),
- branch-and-bound with no-admissible quality function,
- any combination of the above.

This analysis is only a beginning. More work should be done to create good search strategies for hybrid parallel quantum computers for SAT and related problems (see QSPS—chapter 6).

Concluding on SAT realized on QSPS. In theory, every NP problem can be polynomially reduced to SAT. A parallel hybrid Grover-based quantum computer with oracles tuned to solve only SAT problems would be a tremendous asset to all these problems. Here we showed only subset of these problems.

12.5. Oracle for the Exact ESOP Minimization Problem.

12.5.1. Binary Case

In 1988 Martin Helliwell, a PSU student, introduced a decision function for exact ESOP minimization which was later on named the Helliwell Function [Perkowski-Jeske90]. He implemented a GAL-based circuit courtesy Lattice Corporation for hardware minimization of exact ESOPs for single-output 5 variable functions. The generating functions were the all possible products of terms of n-variable function F; there existed thus $N = 3^5 = 243$ of such generating functions. There were 2^5 flip-flops corresponding to every minterm of the function, set initially to the value of a function to be minimized. The problem was to find by an exhaustive hardware search such choice of the generating functions that the EXOR of them would make the states of all flip-flops equal to zero ($F = \sum_g$ iff $F \oplus \sum_g = 0$). A 243 bit binary counter in natural code was used to exhaustively search all combinations of the generating functions so the search was generating worse solutions after already finding a solution with a smaller cost such as generating candidate 00011 after already finding that combination 000011 was a solution. This was the first hardware accelerator for EXOR logic

problem and its performance was much superior to IBM PC AT but the limited size of functions discouraged the PSU team at this time to continue this research. Searching with a binary counter is not a depth-first or a breadth-first method and its only advantage is the simplicity and regularity of hardware.

In 1990 Professor Perkowski and Professor Jeske found several generalizations of the Helliwell's Method to multi-output multiple-valued functions to Positive Polarity Reed-Muller Forms to Fixed Polarity Reed-Muller Forms GRM forms and other [Green91, Perkowski]. The method was implemented in software using depth-first search but unfortunately the limit of 5 variables was not exceeded. However it was observed that the search algorithms can be made much more efficiently for strongly unspecified functions and by using more sophisticated tree search strategies. A tree is pruned by finding equivalent operators on each level.

Now I will formulate the quantum oracle to solve the ESOP Minimization Problem.

Given

(D1) the set of care minterms of a single output function F with the corresponding binary output values of a single output function for each care minterm d.

(D2) The set d of the generating (or basis) functions to be used.

Find

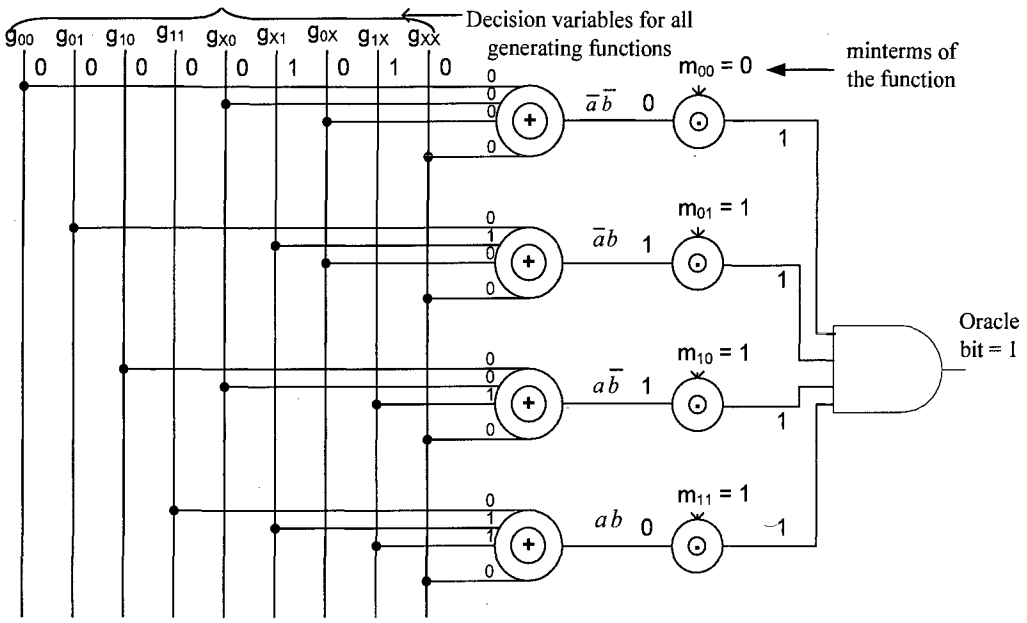
The minimum solution i.e. the expression being an EXOR of generating functions G_i with the minimum number of inputs to the output EXOR gate (i.e. in other words the minimum number of EXORed functions selected from the set of the generating functions from D2).

The algorithm.

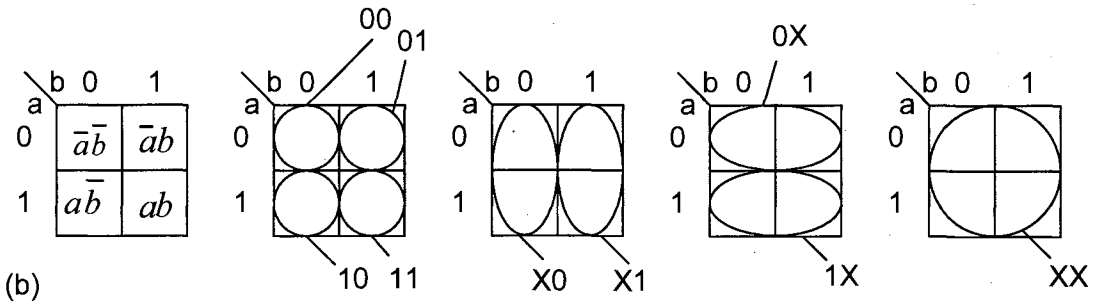
For function F of n variables create an arbitrary number C of all generating functions G_i stored in hypothetical registers $C = 2^n$ for any canonical AND/EXOR form, 3^n for ESOP, $C = 2^n$ for any LI form $C = 3^n$ for non-canonical expressions being generalizations of canonical Maitra LI forms, $C = v * 2^n$ for a combination of generating functions from various canonical forms, LI forms, etc. To every generating function G_i corresponds one binary decision variable g_i in the oracle.

Exoring all selected group variables equals the original function F. The decision function from formula F is a generalization of the Helliwell Function. Its generalization for multi-output case is trivial. The cares of each output must be separately repeated in the vectors. Figure 12.5.1 explains the principle of our approach. This particular example minimizes a completely specified function as an ESOP but very similar oracles can be build for PPRM, FPRM, Maitra, etc, complete and incomplete algorithms. In theory, any method based on LI families can be reduced to these types of oracles.

Create oracle as shown in Figure 12.5.1 and Figure 12.5.2.



(a)



(b)

Figure 12.5.1: Oracle for ESOP to be minimized using the Helliwell's Function. (a) The construction of the oracle for all ESOPs of 2 variables. The first from left level are EXOR gates, the next are EQUIVALENCE gates and the next is the global AND. (b) The minterms of the 2-variable KMap and all generating functions for an ESOP Generating functions are product terms encoded by respective decision variables. For instance, the variable g_{0X} encodes product term $0X = \bar{a}$ and the decision variable g_{00} encodes the product term $00 = \bar{a}\bar{b}$. The circuit in (a) is simulated for $g_{X1} \bullet g_{1X} = 1$,

thus for expression $a \oplus b$ that has value 0 for minterms $\bar{a}\,\bar{b}$ and ab and value 1 for minterms $\bar{a}\,b$ and $a\,\bar{b}$.

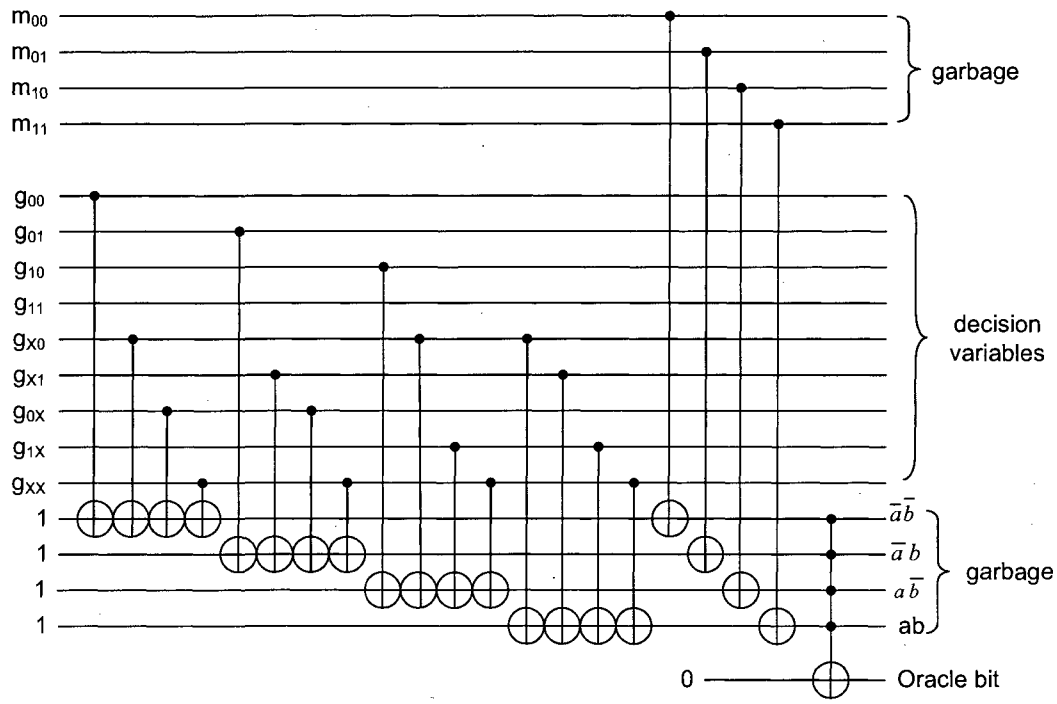


Figure 12.5.2: Quantum Oracle for the oracle from Figure 12.5.1 (Mirror circuit not shown).

Several variants of this algorithm were developed which speedup the operation in some special cases. When no upper bound is known the algorithm with increasing the value of m can be used instead of the above algorithm with decreasing the value of m. In the increasing variant the first solution is the minimum one but usually more iterations are needed. In this variant it speeds the algorithm when we calculate a lower bound of the cost as the starting value of m.

12.5.2. Binary Generalizations.

This section introduced a software/hardware approach to all "even-odd" covering problems using the quantum computer of standard type (a quantum array model) based on Grover. The Boolean decision function to be satisfied with minimum nonzero arguments is a generalization of the Helliwell's function. The incomplete function first simplified in software to disjoint cubes. ON and OFF cubes corresponds now to minterm m_i from Figure 12.5.1 and the oracle reflects the structure of the even-odd covering problem with any generating functions. The search is executed where m is an expected solution cost and N is the number of classes of equivalent generating functions. The method is more efficient when for incomplete functions ***m*** is small even for large ***N.***

Helliwell Function has been used for exact solution to incomplete Exclusive-Or Sum of Products (ESOP) and Fixed Polarity Reed-Muller FPRM forms. Next this method has been extended to other similar problems. Another method investigated for similar applications was the Zakrevskij's Staircase method. Basically, from the deeper theoretical point of view these two methods are the same. We will call them GHF, Generalized Helliwell Function. It can be observed that these methods can be generalized to all problems where the function sought is the canonical form of an EXOR of arbitrary linearly independent (LI) generating functions (chapter 9). Next it can be observed that the EXOR expression can be not necessarily canonical so that arbitrary functions are used instead of LI functions. Finally the methods can be

extended to a non-canonical EXOR expression of arbitrary generating functions. If the matrix of these functions is singular-many (all) solutions are found. If these functions are not Linearly Independent, no solution is found.

Concluding, the presented method is in theory so general that every problem discussed in this thesis in chapters 3, 4, 6, 7, 8, 9 and 10 can be solved. It would require, however, a quantum computer with an exponential number of qubits.

12.5.3. Multiple-valued Generalizations.

Our main idea from section 12.5.1 can be generalized to MV-input binary-output logic (chapter 10) and in case of exhaustive search can be summarized as follows: every multi-output incomplete (multiple-valued) k-nary input, k-nary output function realized in the form of a GF(k) addition of arbitrary functions from a well defined set of functions over GF(k) can be minimized exactly or approximately using quantum oracle for Grover. This may be done in a system that realizes a generalized Helliwell function in the oracle (with hybrid quantum gates as discussed in chapter 10).

Very similar generalized "hybrid" approaches to solving arbitrary hybrid Boolean/MV equations, Generalized Satisfiability Functions, variants of Graph Coloring, MV Maximum Clique Set Covering, MV Petrick Functions and MV Clique Partitioning using oracles can be created, as should be obvious from chapters 10, 11 and 12. In each of them the essence is to perform the enumeration of all subsets and checking

some logical conditions using complete enumeration using quantum superposition. Using a parallel quantum computer various Sequential/parallel Generations that correspond to Depth-First, Breadth-First and other Tree Search methods can be created. Most generally the main contribution of section 12.5 is to propose a very general method to perform arbitrary tree search for NP-complete problems using quantum and parallel quantum computers.

12.6. Conclusion to Chapter 12.

In this chapter, based partially on literature but mostly on our own analysis, I presented a simple and intuitive explanation of basic SAT-like quantum algorithms based on Grover. Using diagrams KMaps, trees and exemplary matrices allowed, I hope to explain these complex subjects in a simple way. Next, I showed SAT family of oracles for very many classical CAD and quantum CAD problems. Figure 12.6.1 presents the reduction graph of just some basic CAD problems. We build quantum oracles for sufficient number of nodes in this graph to be able to solve (in theory) every CAD problem reducible to them. As it can be seen, SAT and graph coloring occupy important place in this graph. So is the maximum clique problems. Some of our oracles have two parts: decision part and cost function minimization, as shown in Figure 12.6.2. Chapters 13-15 will illustrate these and new principles of building oracles for more types of problems for Grover. Reductions for more CAD problems and also for other problems will be given.

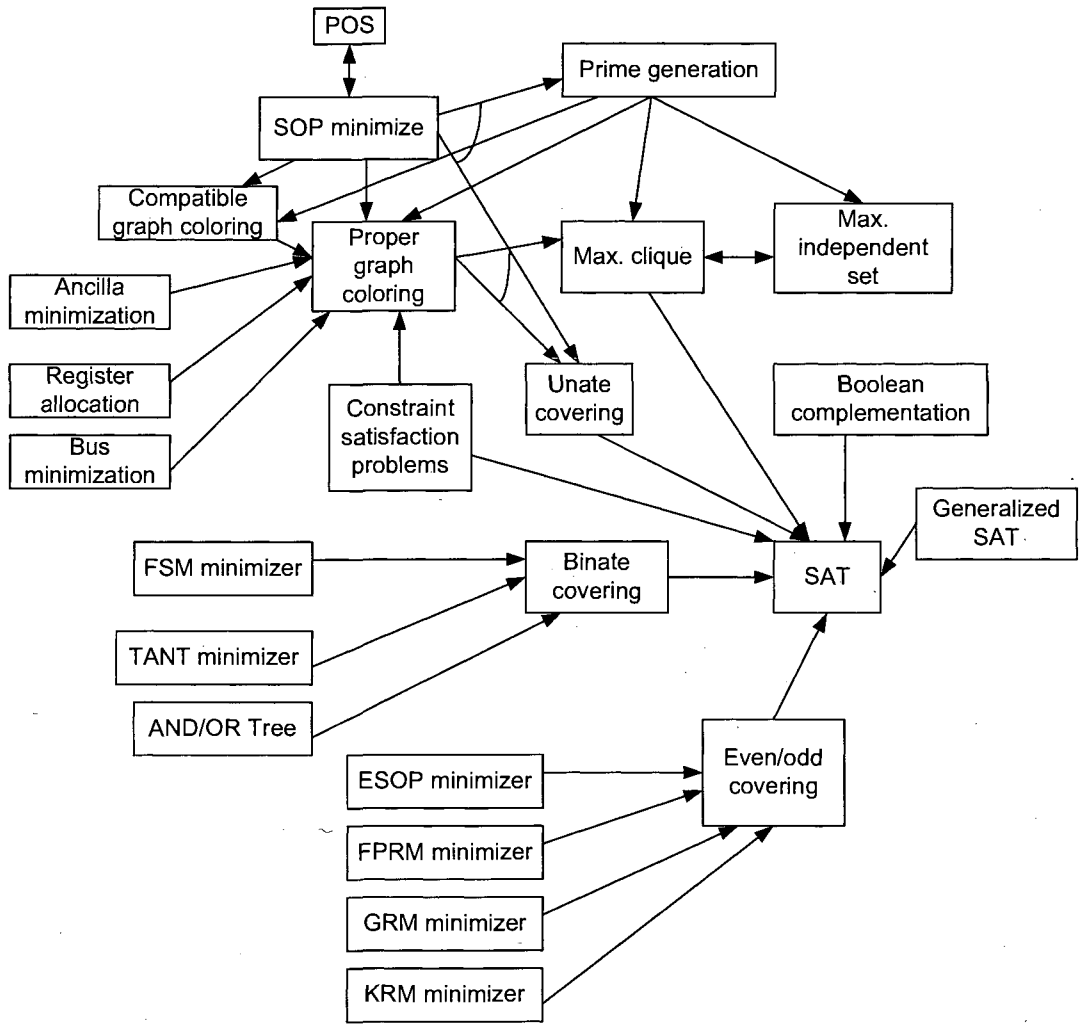


Figure 12.6.1: The reductions of basic CAD and Quantum CAD problems discussed in this dissertation. It should be obvious to the reader familiar with Garey and Johnson seminal book [Garey] that there are hundreds of practical problems efficiently reducible to the problems from this graph, especially to the SAT problem.

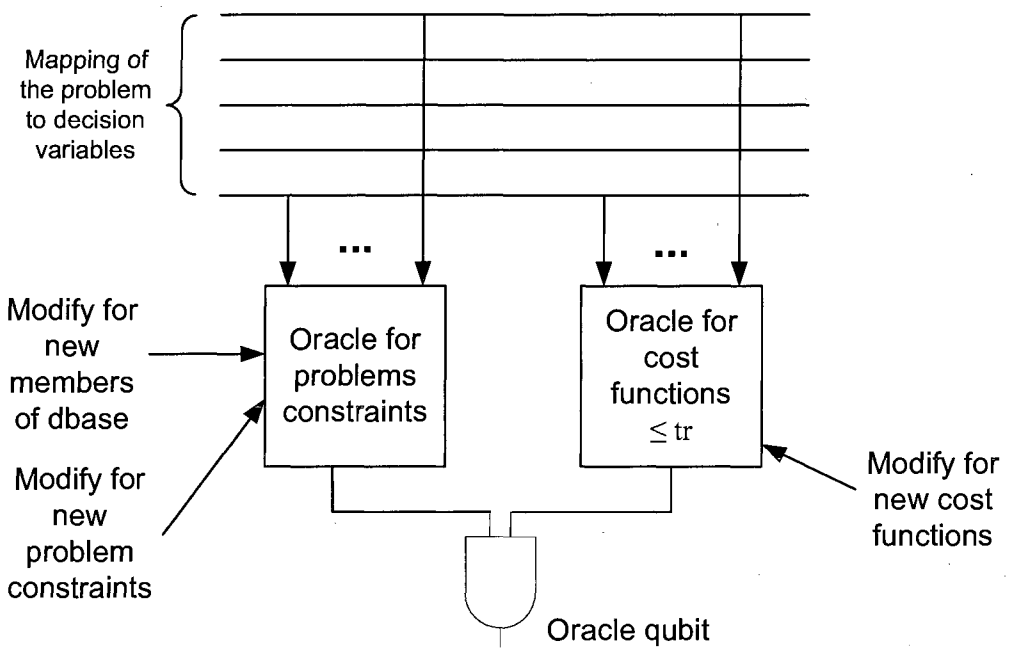


Figure 12.6.2: Schematic representation of Grover oracles for all problems from Figure 12.6.1. Every problem is represented by mapping to decision variables that are given in superposed form to the Grover Loop. The user or an automatic system modifies the decision oracle of the given problem for new instances by modifying the metaphorical "data base" of Grover or a Boolean function model used in this dissertation. The problem itself can be modified by adding or removing some constraints – another redesign of this Boolean function. Finally the optimization part of the oracle is modified by changing the cost function or the way how this cost function is calculated.

CHAPTER 13

Oracle for the Graph Coloring Problems.

13.1. The Graph Coloring Problem.

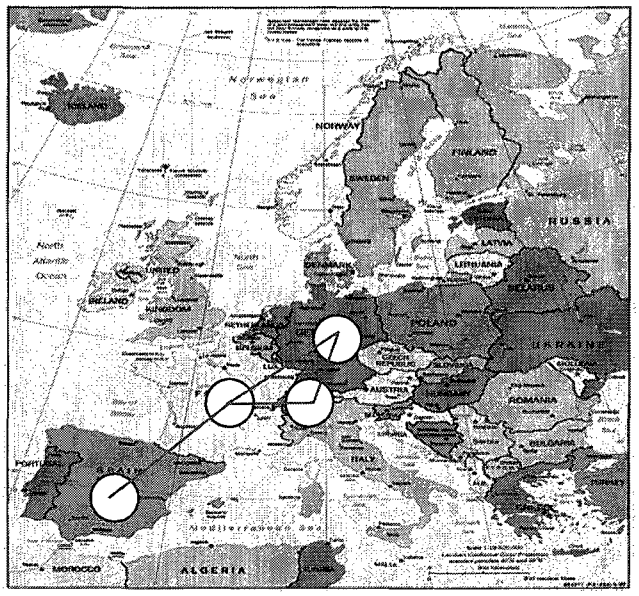


Figure 13.1: Map of Europe.

Graph coloring is a relatively easy problem to formulate in principle, but large amounts of nodes in the graph would result in an extremely large amount of combinations making the problem extremely difficult to solve exactly on a standard computer. Thus this problem is a great candidate for quantum computing. I became interested in the problem of using a quantum computer to find the minimum solution to the graph coloring problem when I found that there is no literature on this subject, although much is known about graph coloring and related problems on standard computers. This gave me the idea to may-be adapting standard graph-coloring approaches in quantum

computing. The main result to be expected from Grover was that the optimal (exact) solution could be found in a number of steps proportional to the square root of N, where N is $2^{n*\log c}$, where n is the number of nodes and c is the upper bound of the number of colors. The classical algorithm would require an order of N amount of steps. Thus, while a classical computer would take 10,000 hours to solve a complex graph coloring problem, the quantum device would take 100 hours. Quantum computing's relative speedup is only quadratic in this case, but in any case it is dramatic in real-life situations like military image recognition. For instance; South Korea installed robots soldiers on its frontier with North Korea that are equipped with image recognition abilities. In this case the quadratic speedup is very important, 5 seconds versus 25 seconds may make a life-or-death difference. There are many other problem instances like this.

In the simplest formulation of graph coloring, a graph is denoted as a standard graph (not a multi-graph) with a certain number of edges and nodes. Every node is connected to at least one other node, by means of an edge. Every node may also obtain a color, which is represented as a bit string. A solution to a graph coloring problem consists of having no uncolored nodes, and having no edges connecting 2 nodes of the same color. We want also to minimize the number of colors used (this leads to finding the chromatic number of the graph). A rather popular branch of graph coloring is called "Map Coloring". Maps, for easy distinction between countries in them, tend to have different, adjacent countries colored differently. For those whose eyesight is not

perfect, the distinction between 2 shapes of different colors is far more easily recognized than a thin black borderline. In graph coloring, each country is represented as a node, and borders are represented as graph edges, (see Figure 13.1). The interest in Map Coloring was started by Francis Guthrie, who in the 1850's formulated a problem involving coloring a map with only 4 colors. The problem remained unsolved until 1976, when after hundreds of computer-hours of calculation, Kenneth Appel and Wolfgang Haken proposed a solution that, as of yet, has not been disproven and mathematicians agree that the solution is correct. Map coloring is thus the first and easy variant of graph coloring and constraint satisfaction problems that I explain and simulate in this thesis. Since it was proved that every map can be colored with 4 colors, our oracle is greatly simplified, especially if one would try to apply it to a very big map.

13.2. Proposed Architecture for Graph Coloring Problem using Grover's Algorithm

In this section, we introduce the proposed architecture for finding the minimum coloring using Grover Algorithm.

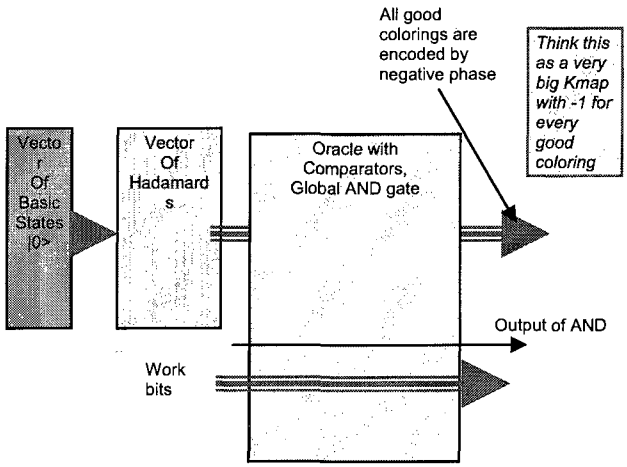


Figure 13.2.1: Block Diagram of creating superposed quantum states with negative phase for all good colorings of a map. Observe that information if a given coloring is good is seen by the output of AND in oracle, but the argument for which the oracle is satisfied is shown in negative phase of the respective minterm of the color encoding variables (recall chapter 4).

Figure 13.2.1 gives the idea of using Grover for graph coloring. Nodes(countries) are represented as groups of neighbor input variables. Coloring of a node is represented as a binary encoding of the set of qubits corresponding to this node. All possible colorings are created at the oracle's inputs by the vector of Hadamard gates on each input. As always, they are all initialized to state $|0\rangle$.

Figure 13.2.2 gives the example of coloring a particular map (left top corner) with inequality comparators and a global AND. The global AND produces a logic one when all neighbor nodes have different nodes. Observe that although the graph is 3-colorable, a coloring with 4 colors is given here as a good coloring because this simple oracle is not trying to minimize the number of colors used for the coloring i.e., (this is a Decision Oracle, not an Optimization Oracle). The first solution out of many can

terminate if the standard Grover algorithm is run. This figure shows also that all primary inputs are repeated to the outputs and forwarded to the next stages together with the output bit(yes/no) of the oracle. The details of Hadamard gates and their initializations are presented in Figure 13.2.3.

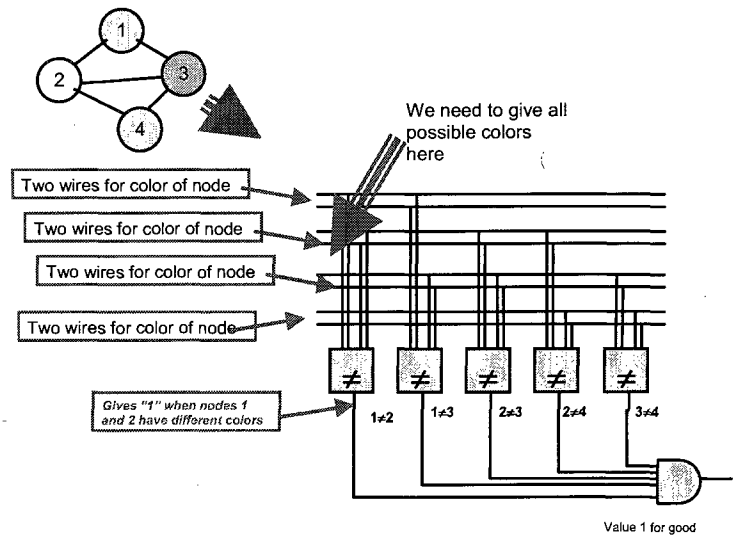


Figure 13.2.2: A simple graph coloring problem: the color comparators correspond to the borders of the countries or the edges of the graph. Observe that this oracle can be used not only in quantum but also in reversible and classical technologies, but in such cases it would require sequential inputs and not parallel superposed inputs as created by Hadamard gates in quantum oracles.

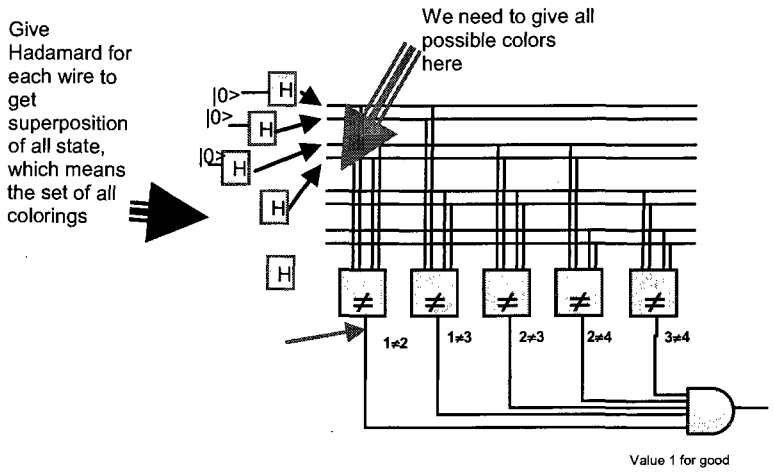


Figure 13.2.3: A simple quantum graph coloring problem: here all the input states are created using zero-initialized Hadamard gates in all variable qubits.

The blocks for the complete Oracle for Graph Coloring and how they are connected together are illustrated in Figure 13.2.4. This oracle is quantum due to the fact that it is comprised solely of quantum gates. Thus all the gates from Figure 13.2.3 are replaced with their quantum counterparts, as discussed in Chapter 11, with all gates build from quantum primitives as discussed in chapters 2 and 3.

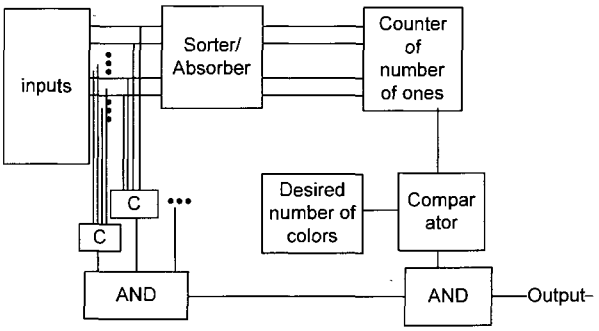


Figure 13.2.4: Simplified schematic of our optimization Graph Coloring Oracle. All blocks have been explained in chapter 11. This oracle is composed from the Decision Oracle on the left (Figure 13.2.6) and color number minimization scheme at the right (Figures 13.2.5 – 13.2.8 and 13.3.1 – 13.3.8), combined with the global AND.

The rough explanation of blocks from Figure 13.2.4 follows.

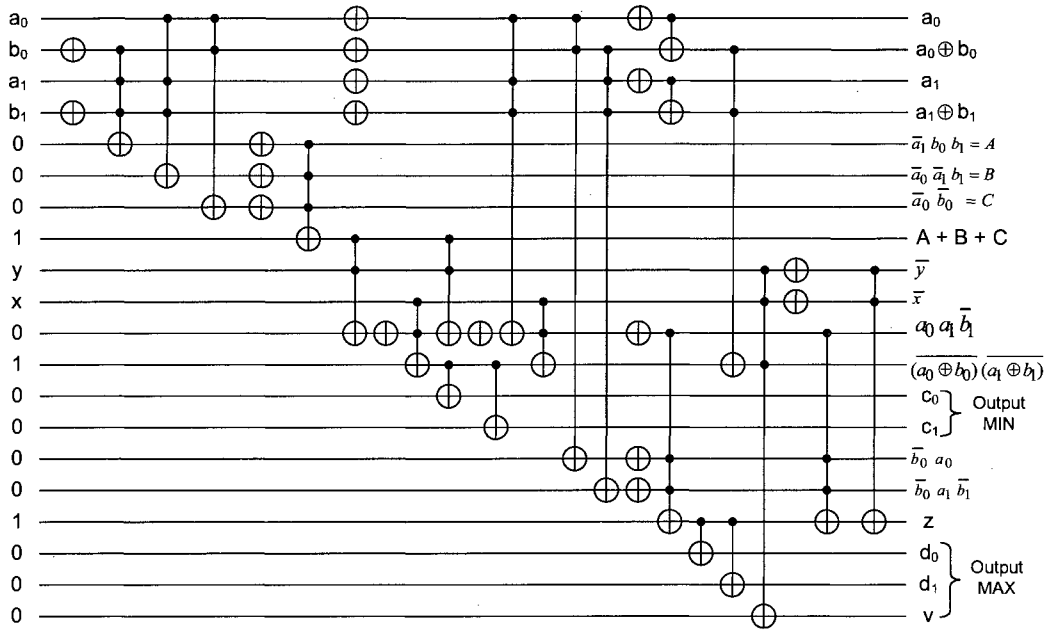
The C blocks:

These are the Inequality Comparators discussed in chapter 3 and chapter 11. As we know, they act upon sets of two inputs. Those two inputs are representative of connected nodes' color encoding. If these two inputs binary strings are the same, then they violated coloring rule and output of the C block will be "0". The quantum oracle is to run through every possible color configuration of inputs (see Figures 13.2.2 and 13.2.3); only a few are solutions. In order to determine whether it is a solution, we run the representative inputs through the comparators. The C comparators outputs are then forwarded into an AND gate at the bottom left to determine whether the configuration is a solution.

The Sorter/Absorber:

Here, the inputted color encodings are sorted. If two inputs are the same for different nodes (same color used more than once), then only one will be outputted and all other same input will be "absorbed" (removed). This circuit sorts and absorbs colors such that all inputs will be sorted from the "smallest" to the "largest" and each color only has one output. We can observe that this is a general circuit to convert a list of items with repetitions to a set with no repeated elements. Again, we designed this circuit in full detail in chapter 11.

The entire circuit is very big and it is difficult to put it on paper. Here we give only some of the blocks and we do not show the complete layout that includes CNOT gates for copying and SWAP gates to be able to combine all blocks together.



(a)

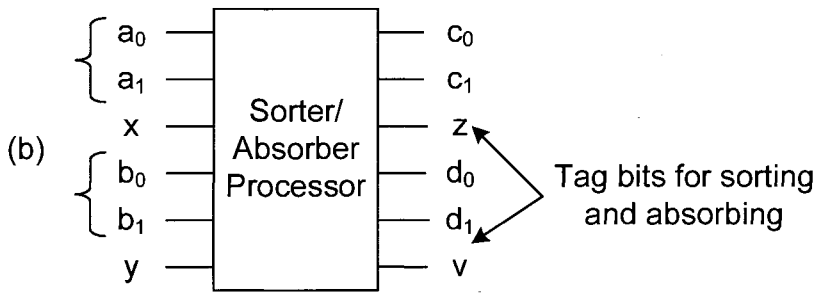
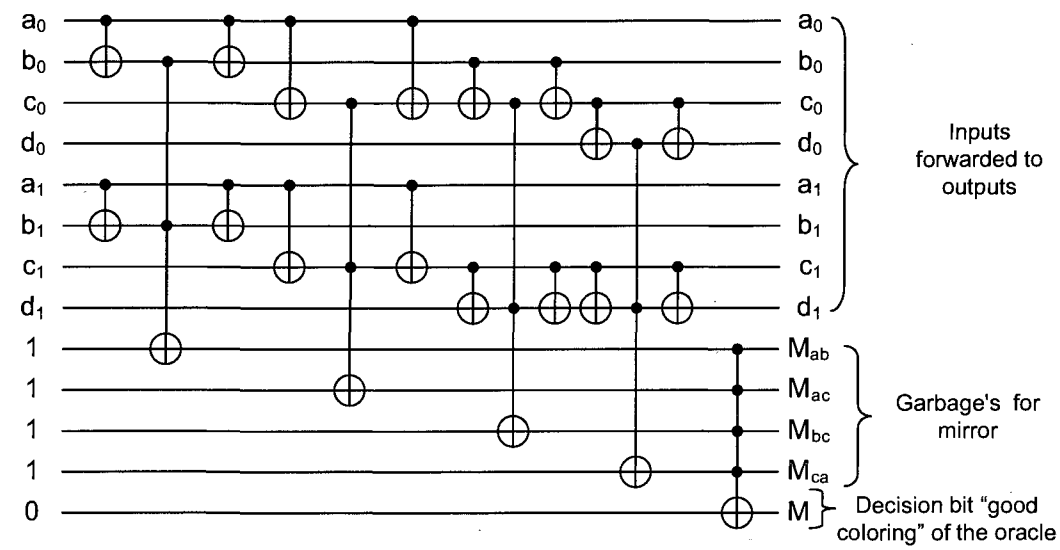


Figure 13.2.5: (a) One block of sorter absorber. We call it sorter/absorber processor. This block is repeated two times in the odd column, one time in the even column, and next these two columns are repeated 2 times. Many mirror circuits are also necessary as will be discussed in section 13.3. Order of inputs a, b should be changed according to the order from oracle. This is done using SWAP gates.(b) The schematics illustrating the use of SWAP gates.



(a)

Figure 13.2.6: (a) Graph coloring oracle – decision part. Order of inputs a, b should be changed according to the order from oracle. This is done using SWAP gates.

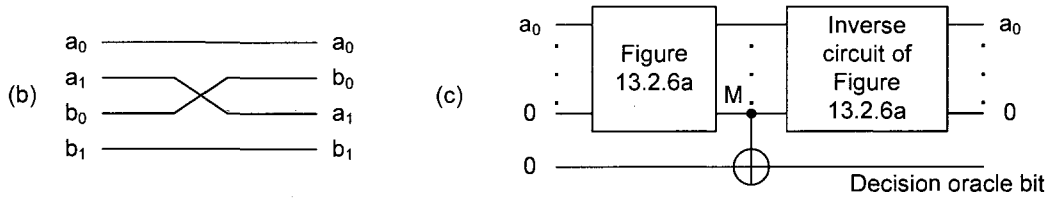


Figure 13.2.6: (b) Preprocessing of the circuit from Figure 13.2.6a using SWAP gates to change order of variables, (c) Inverse circuit-mirror for the decision oracle part.

The color numbers counter:

This counts the number of ones that are in the result that came out of the Sorter/Absorber. The one count can be considered a count of the number of colors. We designed this circuit in Chapter 11 and called it the "Ones Counter". This circuit occurs in most of the optimizing oracles.

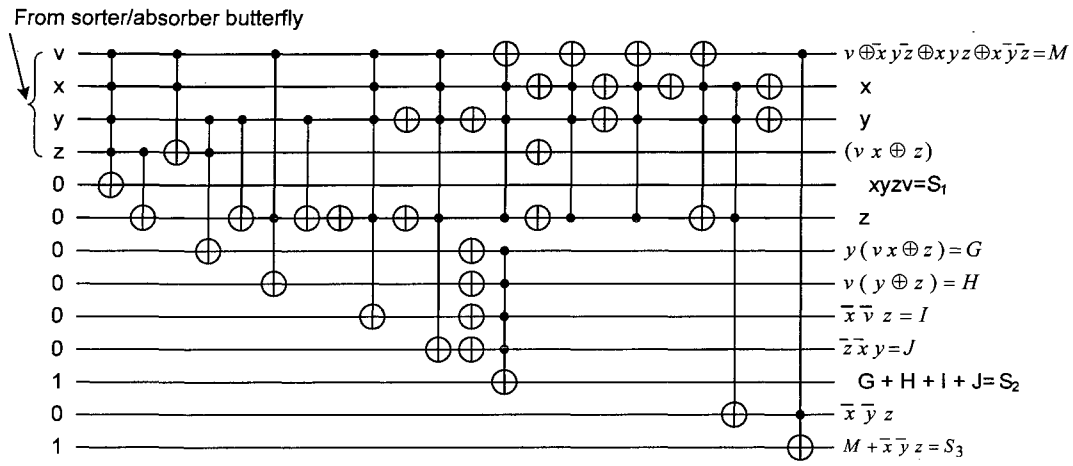


Figure 13.2.7: (a) Graph coloring oracle – counter of ones circuit. Order of inputs x, y, z, v, should be changed according to the order of sort/absorb blocks from sorter/absorber. This is done using SWAP gates.

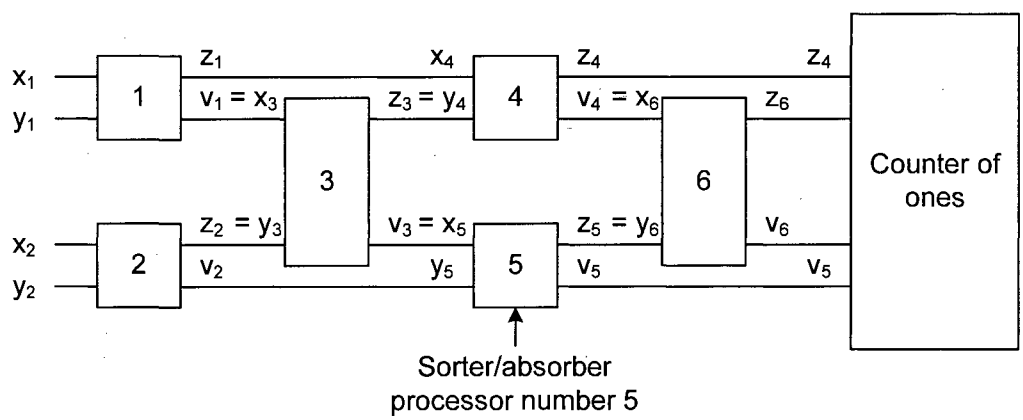


Figure 13.2.7: (b) Explanation of symbols of signals for six blocks of the sorter/absorber butterfly to Figure 13.2.7a.

The Cost Comparator circuit:

This gate acts upon the "number" of colors that was generated by the counter. By using a greater/equal predicate (relation), it can repeatedly compare the input to

desired amounts of numbers to achieve a budget goal. We designed this circuit in Chapter 11. This circuit occurs in all optimizing oracles.

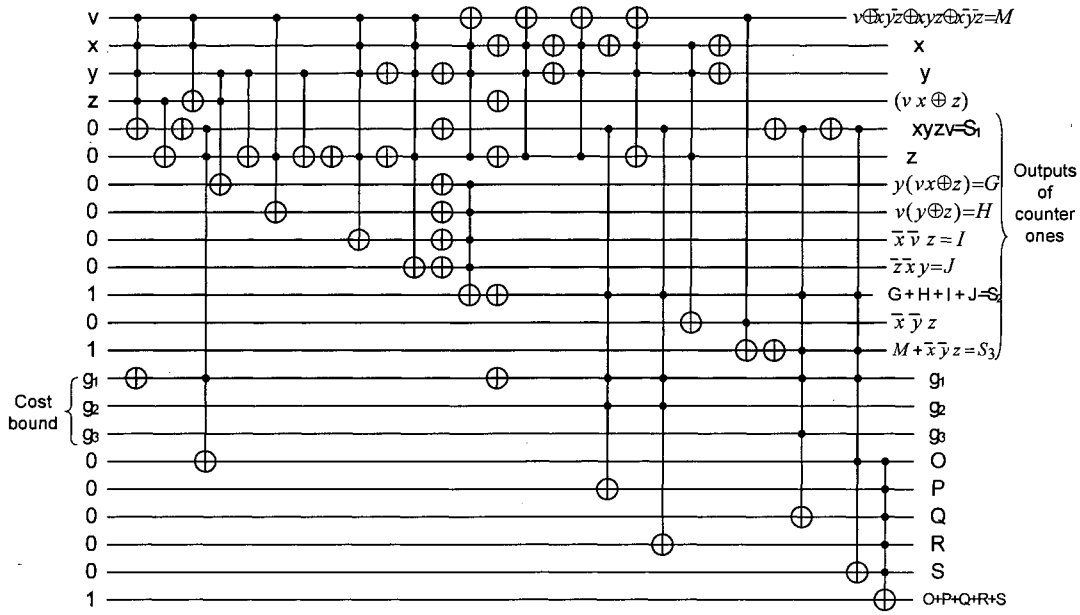


Figure 13.2.8: Graph coloring oracle – complete right part of the oracle optimization part. It includes counter of ones and cost comparator circuits. Order of inputs x, y, z, v, should be changed according to the order of sort/absorb blocks from sorter/absorber. This is done using SWAP gates. The useful qubits are denoted. Other are garbages.

The output of the Cost Comparator Circuit will be AND-ed with "color rule checker" output (output from a big AND gate). This AND gate output is our Oracle output. If it is "1"means that both coloring rules are followed and the number of colors in the configuration is lower than the desired cost threshold (cost bound). This is the search result that we are looking for. If it is zero means that either color rule (no two adjacent notes in same color) is violated, or that the desired color number is not achieved or that both these conditions violated. Then the new coloring arrangement should start

until we get this "oracle" output one, thus solving the problem of "proper" Graph Coloring. The question still remains however, how the inputs are generated. The answer is: all colorings are generated with Hadamard-based superpositions and the desired values are generated in a decreasing order by an external standard computer for which the Grover Algorithm quantum computer is an accelerator.

All these blocks were designed by me already in Chapter 11, so I will not repeat their descriptions here.

13.3. Problems that exist to design the Quantum Layout.

For explanation of the quantum layout problem let us assume four stages of sorter/absorber circuit from figure 13.3.1. Only the question of combining these blocks together is of our interest in section 13.3.

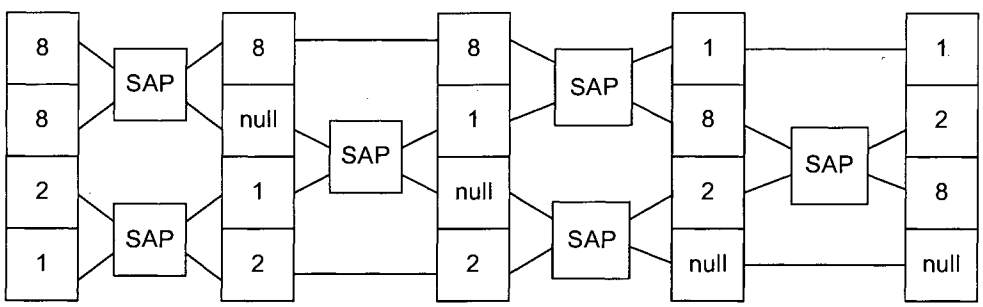


Figure 13.3.1: Butterfly iterative circuit for sorting/absorbing to be used as a single regular block in cost optimizing oracles from Figure 11.9.1 in chapter 11. The registers (rectangles with numbers) in the Data flow Graph are shown for the explanation purpose only. SAP is the sorter/absorber processor.

Layout of the butterfly, which is a completely combinational logic is shown in Figure 13.3.2:

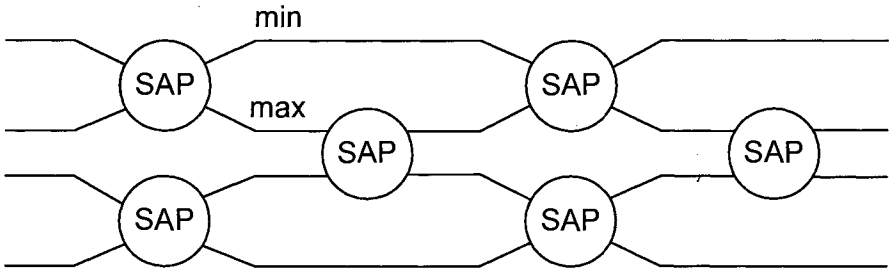


Figure 13.3.2: Butterfly iterative circuit for sorting/absorbing to be used as a block in cost optimizing oracles from Figure 11.9.1 in chapter 11 and in Figure 13.3.1. Circles represent sorting absorbing blocks described as in chapter 11. The reader has to appreciate the regularity of connection patterns in this butterfly combinational logic.

To simplify the explanation of the final quantum layout creation process, we assume that we use a sorting block instead of a sorting/absorbing block and that this is only a one-bit circuit. So MIN becomes AND gate and MAX becomes OR gate in the sorter block.

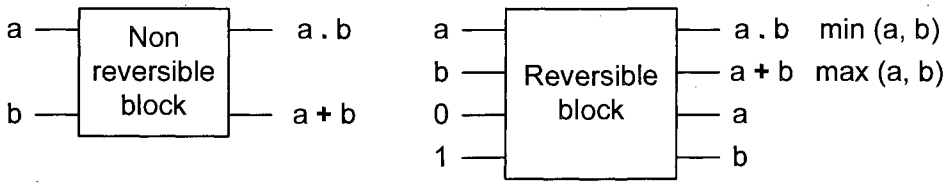


Figure 13.3.3: Single non-reversible block of the Butterfly iterative circuit for sorting/absorbing. External view of a non-reversible and reversible versions of this block to be used in quantum layout of the reversible sorter circuit with mirror circuits.

Figure 13.3.3 presents a simplified non-reversible block of the sorter and next its reversible counterpart. The internals of the block from left in Figure 13.3.3 are redrawn, assuming the width of one bit for every color, to the diagram from Figure 13.3.4. The circuit from Figure 13.3.5 shows classical circuit for the first and second column of the sorter.

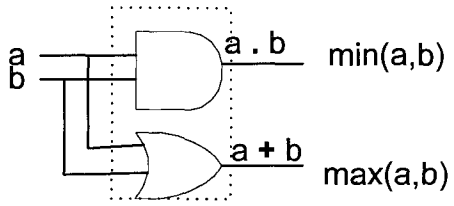


Figure 13.3.4: Single non-reversible block of the Butterfly iterative circuit for sorting/absorbing that shows the internals of the block at left from Figure 13.3.3.

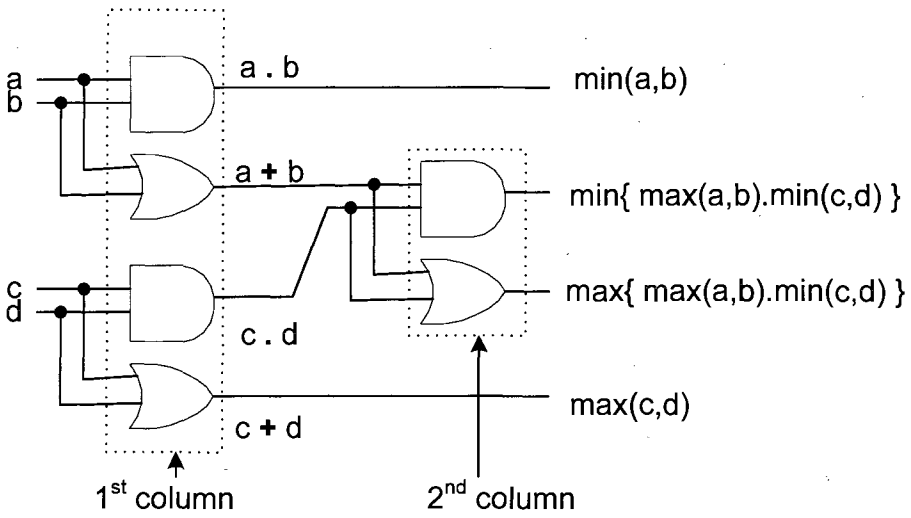


Figure 13.3.5: Three non-reversible blocks of the Butterfly iterative circuit for sorting/absorbing that together correspond to the first and second columns of processors SAP from Figure 13.3.1.

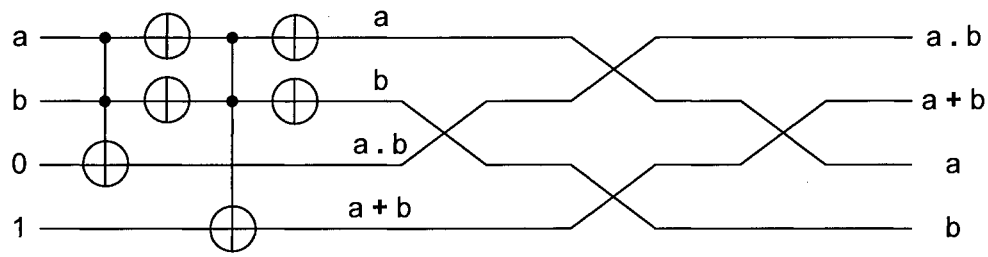


Figure 13.3.6: The single reversible block of the Butterfly iterative circuit for sorting/absorbing with its order of inputs and outputs as required for quantum layout created by adding four SWAP gates at the right.

The final quantum array for the single block of sorter is shown in detail in Figure 13.3.6. Now we can draw the rough schematic of the first three columns of the sorter in block notation – Figure 13.3.7. Mirror circuit for the first two columns is added in Figure 13.3.8.

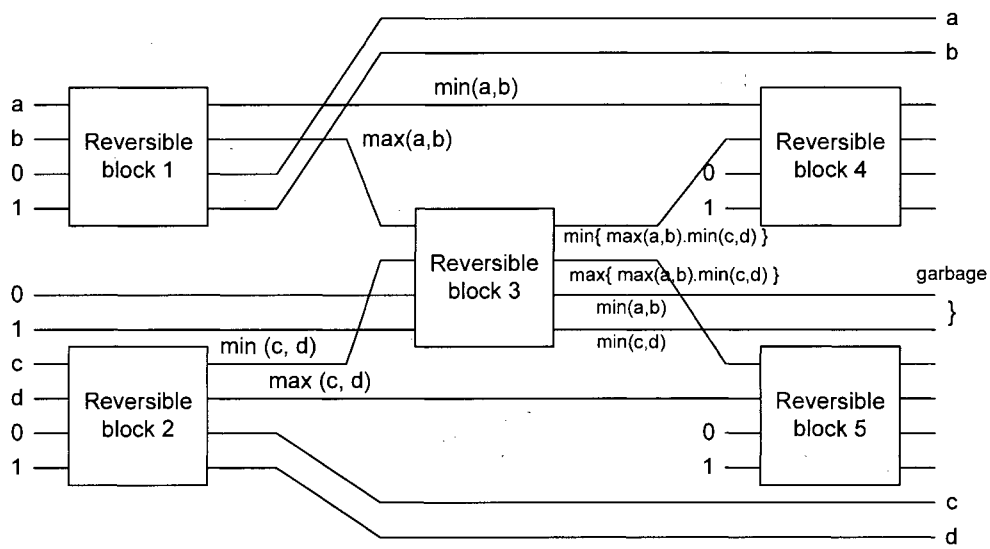


Figure 13.3.7: The block diagram of the first three columns of sorter architecture with its order of inputs and outputs as required for the final quantum layout. It was created by adding SWAP gates, but without final delineation of every qubit of the layout. The mirror circuits of all blocks are also not yet created. Each block's internals should be replaced by the circuit from Figure 13.3.6.

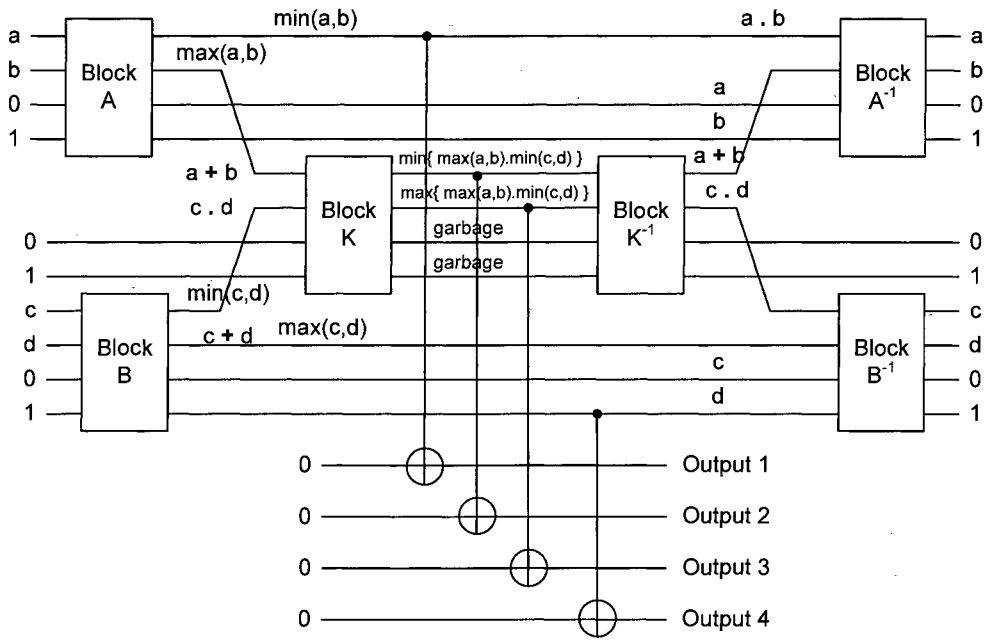


Figure 13.3.8: The final reversible blocks of the Butterfly iterative circuit for sorting/absorbing with 2 columns and with its order of inputs and outputs and mirror circuit. This circuit can be now rewritten to the form of standard quantum array. Each block A, B and K should be replaced by the circuit from Figure 13.3.6. Each block A^{-1}, B^{-1} and K^{-1} should be replaced by the mirror of the circuit from Figure 13.3.6.

The final circuit as a quantum array with 14 qubits can be created by redrawing Figure 13.3.8 to a standard quantum array format with standard notation of SWAP gates and the same distances between any two neighbor qubits. The circuit from Figure 13.3.8 is for simplification drawn for only the first two columns from Figure 13.3.1.

If we replace now the circuit from Figure 13.3.6 with the circuit from Figure 13.2.5 (with added SWAP gates) we will obtain the entire quantum array of the oracle, which is a circuit of a very large size and difficult to draw. This points out to the necessity to create some software that would create, draw and simulate such arrays of large size.

As the next stage we can draw in similar way the complete circuit from Figure 13.2.4 but this results in a very big quantum array diagram. I hope that I presented however the idea of creating quantum layout for multi-level (iterative) circuits by adding mirrors, SWAP gates and Feynman gates.

CHAPTER 14

Oracles for Constraint Satisfaction Problems

Constraints Satisfaction Problems have many applications in computer science, physics, engineering, astronomy, biology and other areas. The problem is formulated by a set of constraints and a cost function.

The problem is formulated as a graph G = <NO, ED> with NO being a set of nodes and ED $\subset$ NO x NO x ... NO being a set of constraints. Constraints can be on any subset of nodes from NO. The nodes can have values such as symbolic or numeric. Any node can have some set of values V(Ni). The simplest constraints are edges from NO x NO. The constraints can be of any type, for instance EQUAL (N1, N3), NOT-EQUAL (N2, N5), SMALLER-THAN (N3, N0).

There are two formulations: ***Constraints_Only*** and ***Constraints_And_Cost_Function***.

Problem 1. Given is Graph G of constraints. Find such assignment of values to variables that all constraints are satisfied.

Problem 2. Given is Graph G of constraints. Given is cost function defined on G with integer of real values. Find such assignment of values to variables that all constraints are satisfied and the cost function is maximized.

In this chapter we will show few applications of Constraint Satisfaction.

14.1. Constraints Satisfaction Problems that are also Equational Logic Problems.

Many Constraint Satisfaction problems can be reduced to a set of logic equations and next to a single equation. This idea comes from Raymon Lullus who lived in thirteen century and was next generalized and formalized by Descartes and finally applied to "Boolean data" by George Boole. The operators in these equations can be of many types such as: arithmetic (+,*,/,-, etc), relational (predicates <. >, = , ≤, ≠, ≥, etc) and logic (AND, OR, EXOR, etc). The cryptographic puzzle belong to this category of problems. SEND + MORE = MONEY is the famous cryptographical puzzle—see Figure 14.1.1. The letters should be replaced with unique digits 0,....9 to make the equation valid. Directly from Figure 14.1.1 one can compile the Equation from Figure 14.1.2.

```
  S E N D
+ M O R E
---------
M O N E Y
```

Figure 14.1.1: Cryptographic problem example. Substitute digits for letters to make the equation to be true.

$$D + E = 10\,C_1 + Y \qquad C_1 \in \{0, 1\}$$

$$N + R + C_1 = 10\,C_2 + E \qquad C_2 \in \{0, 1\}$$

$$E + O + C_2 = 10\,C_3 + N \qquad C_3 \in \{0, 1\}$$

$$S + M + C_3 = 10\,C_4 + O \qquad C_4 \in \{0, 1\}$$

$$C_4 = M$$

Figure 14.1.2: Equations compiled from the problem formulation from Figure 14.1.1.

The specification of nodes is given in Figure 14.1.3. Observe that the carries C_i are binary single-qubit signals but all letters require four qubits in binary encoding, as shown in Figure 14.1.3. This Figure explains also that only some 4-bit strings are allowed, namely the strings 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001.

$S \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$E \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$N \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$D \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$M \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$O \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$R \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$Y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Figure 14.1.3: Constraints for nodes in the graph. Each node is a 4-qubit string. Because we need 4 qubits to represent letters S, E, ..., Y, we need additional constraint to restrict the domain to digits.

$S \neq E, S \neq N, S \neq D, S \neq M$ etc.

Figure 14.1.4: Inequalities for unique encoding of nodes of the graph. One inequality is created for every pair of letters.

The equations from Figure 14.1.2 correspond directly to the rules of arithmetic addition with carry. Carry signals are denoted by C_1, C_2 , C_3 and C_4 ; $C_i \in \{ 0 , 1 \}$. The equations in Figure 14.1.3 state that each symbol S, E,.....Y is a digit from 0 to 9. The equations in Figure 14.1.4 mean that all mappings of symbols S, E, ...Y to digits are unique, i.e. that they are one-to-one mappings. These are all typical equations that lead to typical arithmetic, predicate, logic circuits for a wide class of problems.

$C_4 = M$ $\qquad\qquad$ $C_4 \in \{ 0, 1 \}$

$M = 1$

$S + M + C_3 = 10\, C_4 + O$

$S + M + C_3 = 10\, M + O$

$S + C_3 = 9\, M + O$

$S + C_3 = 9 + O$

Figure 14.1.5: Simplified Equations compiled from Figure 14.1.2.

Equations from Figure 14.1.2 can be simplified to the form from Figure 14.1.5. This would simplify the oracle and speed-up the Grover Algorithm but we will not discuss this "intelligent preprocessing" variant here.

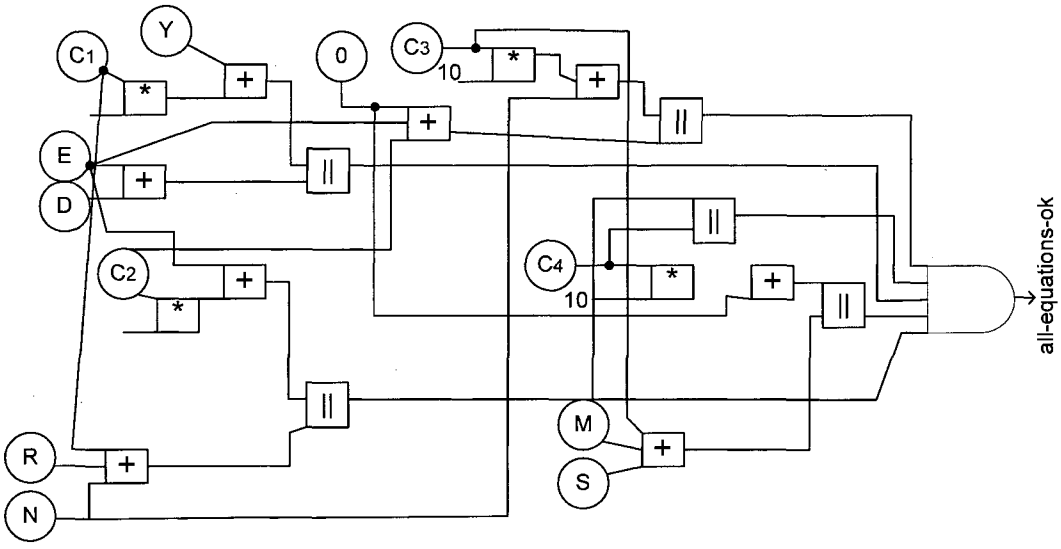


Figure 14.1.6: Graph of constraints for the SEND+MORE=MONEY problem.

Figure 14.1.6 presents the part of the oracle to verify the equations from Figure 14.1.2. The operations of addition, multiplication and equality checking are replaced by logic blocks. Every variable S, E, ...Y has 4 bits. The global AND has output 1 when all equations from Figure 14.1.2 are satisfied. The output of this AND gate is denoted by all-equations-ok. As we see in Figure 14.1.6 we use only the following blocks: arithmetic adder with 2 inputs, arithmetic adder with 3 inputs, multiplier by 1 and 0, equality comparator and AND. Chapter 11 shows how to design all these blocks.

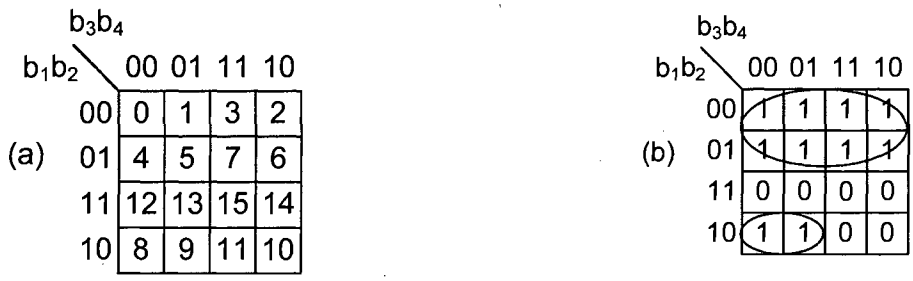


Figure 14.1.7: (a) Enumeration of cells in the M-map, (b)Groups of true minterms in the KMap for the circuit to check each equation from Figure 14.1.3.

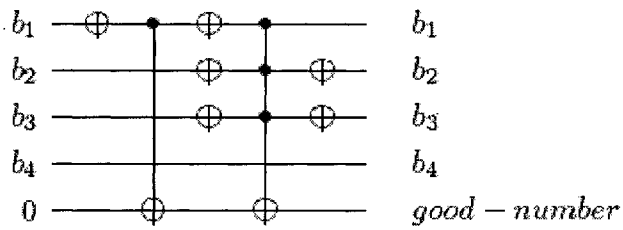


Figure 14.1.8: Realization of circuit GN that checks if an argument is a binary-encoded digit, i.e. that checks if the binary argument is a Good Number, i.e., a digit 0,..., 9.

Figure 14.1.7a presents the method to calculate the circuit to verify that the argument $b_1b_2b_3b_4$ is a binary encoding of a digit 0,.....9. Figure 14.1.7a presents the numbers of cells of KMap — all cells with values 0,....9 have output 1. This leads to the KMap will loops $\bar{b}_1$ and $b_1\bar{b}_2\bar{b}_3$ from Figure 14.1.7b and finally to the circuit from Figure 14.1.8 being a part of the oracle GN. When the binary input combination $b_1b_2b_3b_4$ corresponds to a digit 0,.....9 then the output good-number = 1.

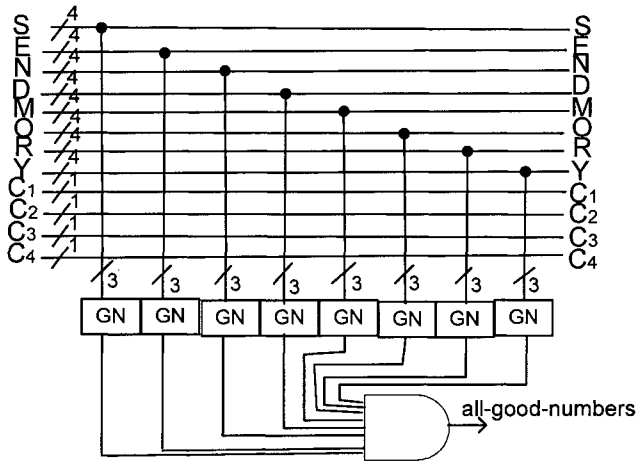
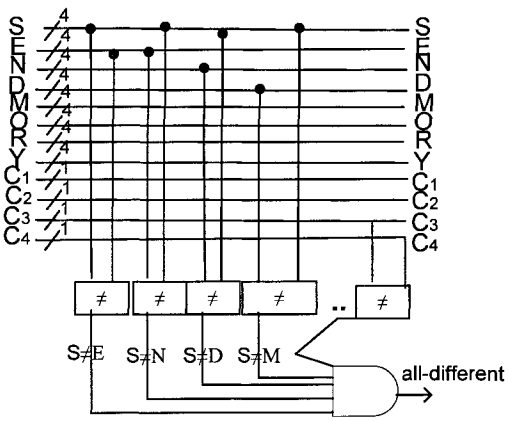


Figure 14.1.9: The remaining part of the oracle All-Good-Number for the SEND+MORE=MONEY problem. This checks the encoding of each symbol S, E, ..., Y. It has 8 GN blocks from Figure 14.1.8 and the global AND.

The part of the oracle that checks all numbers used in equations from Figure 14.1.2 is shown in Figure 14.1.9. GN is the block from Figure 14.1.8. Each such block uses only 3 qubits out of 4 qubits encoding every symbol S, E, ..., Y. This is marked with symbol "3" in vertical buses on inputs to each block GN in Figure 14.1.9. The output of this sub-oracle is denoted by all-good-numbers. All equations from Figure 14.1.3 are verified in the sub-oracle from Figure 14.1.10. The AND gate produces the signal all-different = 1 meaning that the mapping is a one-to-one mapping. The circuits from Figure 14.1.9 and Figure 14.1.10 are typical for many oracles for extended logic equations.

(a)



(b)

SE	SN	SD	SM	SO	SR	SY
	EN	ED	EM	EO	ER	EY
		ND	NM	NO	NR	NY
			DM	DO	DR	DY
				MO	MR	MY
					OR	OY
						RY

Figure 14.1.10: (a) The part of an oracle All-Different for the SEND+MORE=MONEY problem that checks if the mapping is a one-to-one mapping, (b)systematic method to create all pairs of symbols for pair wise comparisons.

Finally, Figure 14.1.11 shows the entire oracle for the SEND + MORE = MONEY problem that is composed of 3 oracles. The final global AND is the logic AND (conjunction) of answers from the partial oracles:

$$solution = all - equations - ok \bullet all - good - numbers \bullet all - different$$

We just need a single 4×4 Toffoli gate with target bit initialized to 0 to realize this final global AND, see at the bottom of Figure 14.1.11.

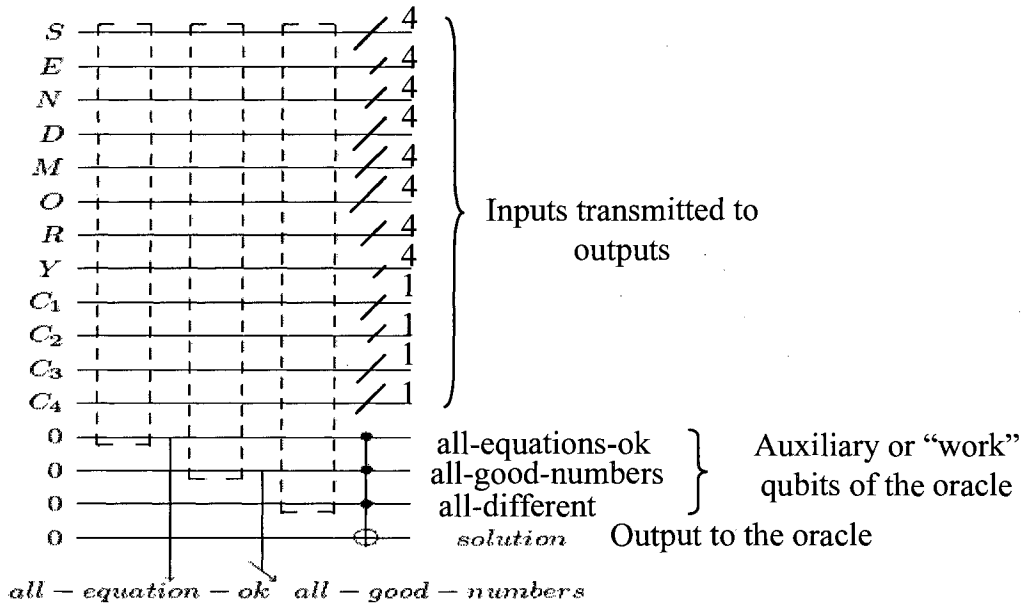


Figure 14.1.11: The complete quantum oracle for the SEND+MORE=MONEY problem. The output is the "solution" qubit at the bottom.

The complete detailed quantum array for the all-good-numbers predicate is given in Figure 14.1.12. It is the checker of All-Good-Numbers from 8 GN blocks. Similarly the whole oracle and HZH circuit is designed.

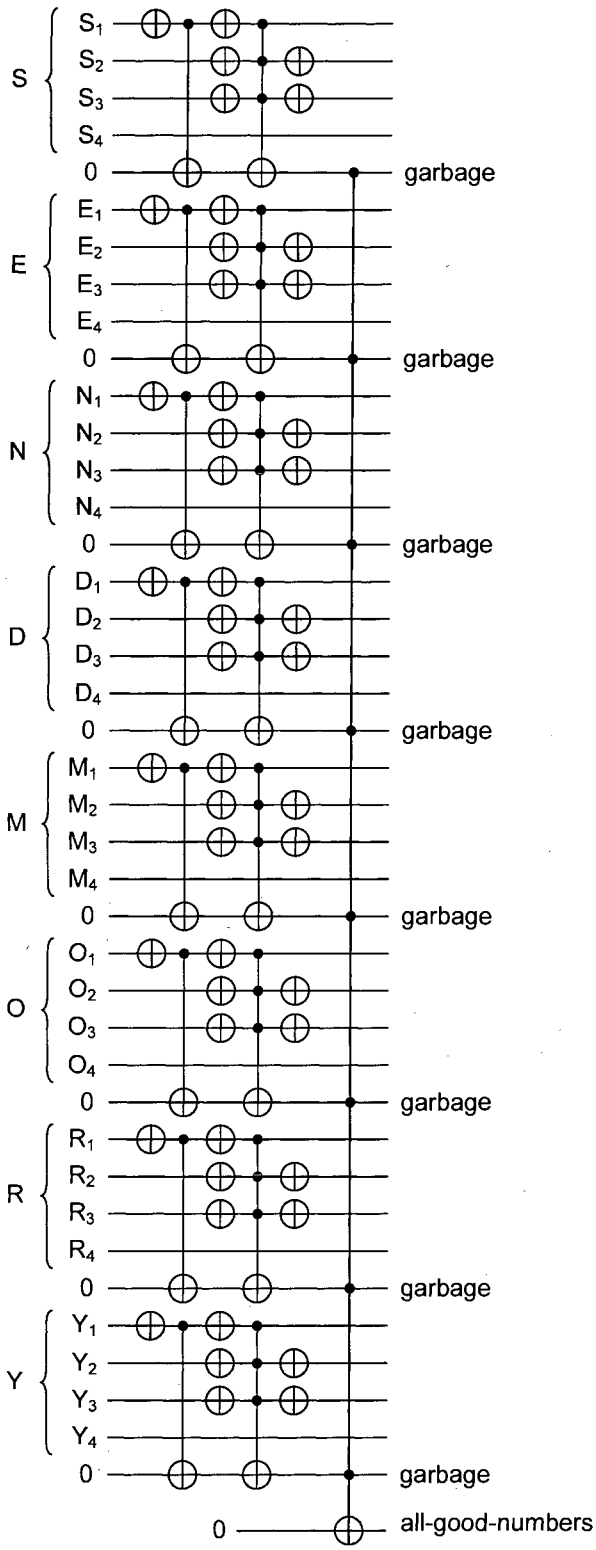


Figure 14.1.12: The part of oracle to calculate the all-good-numbers predicate.

CHAPTER 15

Towards Grover-Based Parallel Quantum Computers for Robotics and Adiabatic Quantum Computing

15.1. Introduction.

In previous chapters we discussed oracles for various applications but these were mostly toy problems (like the SEND+MORE=MONEY problem) or problems of classical and quantum CAD. It should be however observed that the general problem formulations based on oracles, such as satisfiability, mapping problems, path problems and constraint satisfaction problems occur in many other areas. Because I am interested in teaching robotics when I will return to Bangladesh where I am a professor, I looked to potential applications of quantum computers in robotics.

15.2. Constraint Satisfaction Model for Robotics.

Based on literature and what I learned from my PSU experience and robotics classes, one weakness of contemporary robotics is the insufficient speed of robot's image processing, pattern recognition, reasoning and motion planning algorithms. Also in other areas related to perception and reasoning the contemporary computers are just too slow for both the requirements and the existing mechanical abilities of modern robots. This problem can be solved by using special processors which are usually

multiprocessors, and thus expensive and difficult to use. Another approach is using Digital Signal Processors (DSP processors) which have applications especially in image and sonar processing, sometimes also in intelligent motion planning and generation. Finally, highly parallel, sometimes dynamic (adaptable) Field Programmable Gate Array (FPGA) architectures are also used in robotics, and especially in robot vision. The PSU group experimented already with some of these approaches in their past research and found them difficult to use and restricted in applications. The trouble is that designing parallel systems or programming the multi-processing or DSP algorithms is very time consuming. On the other hand, it is well-known that there exist the concepts of the "universal problem solvers"; as an example one can give "automatic theorem proving" programs based on resolution, or logic programming languages such as Prolog. They find applications in CSP. These universal problem solvers allow to write all kinds of such highly complex rule-based hierarchical search programs very quickly, but their practical applications are limited because they just run too long on contemporary computers. It is still fascinating to be able to formulate and solve many different problems using the same general model. This model may be predicate calculus, Satisfiability, Artificial Neural Nets or the Constraints Satisfaction Model. We showed in chapters 12 and 13 that Grover algorithms with the ability of designing oracles for this algorithm are in a sense such a "universal problem-solving algorithm in the quantum world". This approach, as illustrated in chapter 6 has also a natural synergy with parallel processing. In previous

chapters we showed thus two strongly interrelated general purpose problem-solving models:

1. the general search on a standard (serial architecture) computer and on a parallel computer (using any parallel architecture, such as pipelined processor, systolic processor, Single Instruction Multiple-Data architecture (SIMD), etc).
2. the quantum search algorithm by Grover with user-designed problem-specific oracles.

These two models can be combined when a high-level standard computer with search algorithm calls many quantum accelerators for specific sub-problems to be solved independently (possibly in parallel) with Grover-like speedup each.

What may be then the general purpose model for robotics? It is well-known from published robotic research that many known and practical algorithms, for instance the well-known "Waltz algorithm" for "blocks world model vision" (and its derivative algorithms) can be reduced to the general purpose constraint satisfaction problem which in turn can be reduced to the generalized satisfiability problem. For instance, Huffman and Clowes created an approach to polyhedral scene analysis, scenes with opaque, trihedral solids, next improved significantly by Waltz [Waltz75], which popularized the concept of constraints satisfaction and its use in problem solving, especially in image interpretation. Objects in this approach had always three plane

surfaces intersecting in every vertex. Thus there are 18 possible trihedral vertices in this problem out of 64 possible.

There are only three types of edges that are possible between these blocks:

1. Obscuring edge is a boundary between objects or objects and background. Boundary lines are found using outlines with no outside vertices,
2. Concave edges are edges between two object's faces forming an acute angle when seen from outside,
3. Convex edges are those between two faces of an object forming an obtuse angle as seen from outside.

There are only four ways to label a line in this Blocks World Model. The line can be convex, concave, a boundary line facing up and a boundary line facing down (left, or right). The direction of the boundary line depends on the side of the line corresponding to the face of the causing it object. Waltz created the famous algorithm which for this world model always finds the unique correct labeling if a figure is correct. Moreover, the algorithm handled also shadows and cracks in blocks. Mackworth and Sugihara extended this work to arbitrary polyhedra and Malik extended it to smooth curved objects. The extended approach becomes a well-known approach to image recognition based on constraint satisfaction and a prototype of many similar approaches to vision and planning problems in robotics.

Waltz algorithm is an example of constraints satisfaction and the Constraint Satisfaction Model is one of few fundamental models used in robotics [Beach03, Minton90, Fromherz01, Gualandi04, Huang01, Pai96]. Constraint Satisfaction is used in main areas of robotics and especially in vision, knowledge acquisition, knowledge usage, etc., including in particular the following:

- planning, including motion planning, gesture planning, assembly planning, spatial and temporal planning for robot groups, experiment planning,
- scheduling, combined planning and scheduling, multi-robot task planning and scheduling,
- allocation, including resource allocation in AI, graph theoretical problem formulations of robotic problems including graph coloring, graph matching, floor-plan design,
- temporal reasoning,
- assignment and mapping problems,
- arc and path consistency,
- general matching problems,
- belief maintenance,
- satisfiability and Boolean/mixed equation solving,
- machine design and manufacturing,
- diagnostic reasoning,
- qualitative and symbolic reasoning,

- decision support,
- computational linguistics,
- hardware design and hardware verification for robotic applications, configuration of robot systems and factory automation systems,
- real-time systems related to robot planning,
- implementation of non-conflicting sensor systems,
- man-robot and robot-robot communication systems and protocols,
- contingency-tolerant motion control, multi-robot motion planning, ,
- coordination of a group of robots,
- and many others.

Oracles for some of the above problems are either identical or similar to those discussed by us in Chapters 12, 13 and 14. Universal components for these and other algorithms were presented in Chapter 11. We created thus a general approach to solve many of these problems. Moreover our approach can be applied to all constraint satifaction problems, at least in theory, as they can be all reduced to satisfiability, as known from Garey and Johnson [Garey79]. There are however better ways than reduce everything to SAT. For instance the "robot guard problem" is the problem of placing the minimum number of robot guards to watch certain territory of a given shape. This problem is reduced to the unate set covering problem from Chapter 6 for which we built a quantum oracle in Chapter 12. As

another example, the problem of robot scheduling can be reduced to the binate covering problem with costs, also discussed in Chapter 12 of this dissertation.

Now we will rephrase the main methodology of this thesis from the robotics point of view:

1. Reduce robotic problems that need speed to the problem of building a quantum oracle, possibly using a unified constraint satisfaction framework. (Because of fundamental role of basic combinatorial problems, this step can be applied to both CAD and robotics problems.)
2. If there exists a quantum computer based on the "classical quantum circuit model" (which we so far assumed to exist in this thesis) then use this computer to solve the problem.
3. If there is a quantum adiabatic computer available, reduce the problem from the quantum circuit model to the adiabatic quantum model and solve it using the adiabatic quantum model. The match of the constraints satisfaction and the adiabatic quantum computing seems to be perfect. This synergy determines thus the future area of quantum robotics, at least in the coming years, because as of 2008 very likely adiabatic quantum computing will be available first.

15.3. Adiabatic Quantum Computing to Solve Constraint Satisfaction Problems Efficiently.

It is quite possible that the date of February 13th 2007 will be remembered in annals of computing. DWAVE Company demonstrated their 16-qubit Orion quantum computing system in Computer History Museum in Mountain View, California. It was the first time in history that a commercial quantum computer was presented, although it was only a prototype model, needed scaling up, and there is also a doubt among some researchers if the computer really gives the quadratic speedup. On November 27, 2007 a 28-qubit Orion was demonstrated. The Orion system is a hardware accelerator designed to solve in principle a particular NP-complete problem called the two-dimensional Ising model in a magnetic field (for instance quadratic programming). It is built around a 28-qubit superconducting adiabatic quantum computer (AQC) processor. The system is designed to be used together with a conventional front-end for any application that requires the solution of an NP-complete problem. The first application that was demonstrated was pattern matching applied to searching databases of molecules. The second was a planning/scheduling application for assigning people to seats subject to constraints. This is an example of applying Orion to constraint satisfaction problems. The third was Sudoku. The company promises to provide free access by Internet in 2008 to one of their systems to those researchers who want to develop their own applications.

The plans in 2007 were that by the end of year 2008 the Orion systems will be scaled to more than 1000 qubits. It is even more amazing that the company plans to build in 2009 new processors specifically designed for quantum simulation, which represents a huge commercial opportunity. Interesting information can be found on the company's webpage. These problems include protein folding, drug design and many other in chemistry, biology and material science. Thus the company attempts to dominate enormous markets of NP-complete problems and quantum simulation. If successful, the arrival of adiabatic quantum computers will create a need for the development of new algorithms and adaptations of existing search algorithms (quantum or not) for the DWAVE architecture. The arrival of Orion systems is certainly an excellent news for any research group that is interested in formulating problems to be solved on a quantum computer. I hope that in forthcoming projects some next Ph.D students at PSU will concentrate on robotic applications of the Constraint Satisfaction Model and will use the ORION computer according to the method specified below.

Adiabatic Quantum Computing was proved equivalent [Aharonov03, Mizel07] to standard QC circuit model that we illustrated in previous chapters and used in [Bae07, Giesecke07, Giesecke06, Hung06, Khan06, Khan05a, Khan05b, Kumer07, Lee06, Lukac07, Lukac07a, Lukac07b, Li06, Perkowski05, Perkowski07a, Perkowski07b, Raghuvanshi07, Song06, Yang06, Yang05e]. Therefore, at least in theory, each of the developed by us oracles together with the Grover's Algorithm problem-independent circuits in the Grover Loop create together a very large quantum circuit that in

principle can be transformed to an equivalent adiabatic quantum program and run on the Orion computer. In previous chapters we developed both: the algorithms for problems and the methods to design oracles for them. Thus the final description can be created by hand. I hope that in future some PSU students will develop automatic software for "quantum layout" to compile a composition of small circuits to one big circuit and its matrix.

We want to solve at first the relatively simple problems such as Maximum Clique or SAT. This programming would be now like on the "assembly level" or "machine language" but with time more efficient methods will be developed in the PSU quantum group. DWAVE gives API in XML as an interface for remote running of their computer. This is conceptually similar to programming the contemporary Field-Programmable Gate Arrays. The processor is programmable for a particular graph abstracting the problem. I think that one can safely predict that in future the adaptations of many methods developed for FPGAs will be used for quantum computers, including the adiabatic quantum computers.

Several aspects presented below should be used in further research and can be considered while creating "software API" for the Orion AQC:

15.3.1. One method of creating software for AQC is by formulating an oracle for Grover algorithm and next converting it to the AQC model [Aharonov03, Mizel07]. As discussed in previous two chapters, the quantum oracle is a

quantum permutative circuit that has a mapping given to oracle's input qubits. The oracle answers only yes/no at its output. For instance, building a graph-coloring quantum computer requires constructing an oracle that gives answers only like this: "this mapping of nodes to colors is a proper coloring" while a proper coloring is one that every neighbor nodes are mapped to different colors. Another quantum oracle may answer "this coloring is proper and the number of colors used is smaller than 5". Designing practical oracles for Grover algorithm [Li06] is not a well researched area yet and this dissertation is the first that tries to contribute to it, but the interface to DWAVE is not yet completed. Oracles for famous fundamental NP problems in robotics, CAD and other areas should be built to practically evaluate the synthesis methods that are known or proposed in this thesis. Building an oracle requires the ability to synthesize a complex permutative circuit from universal binary gates such as Toffoli or Fredkin [Lukac03] and new gates, such as the affine gates proposed in this dissertation. It helps also to know and reuse standard quantum logic blocks (see chapter 11 and [Khan05a, Khan05c]).

15.3.2. The Adiabatic equivalent of Grover algorithm is implemented in Orion system and Hamiltonians for 16-qubit oracles can be built for the Orion system. Twenty eight qubits is still a "toy problem" for some problems and is not enough for many practical robot-related constraint satisfaction

problems. It is however a good starting point for self-education, software development and to prove a point of quantum computing. The created in our group minimization methods (chapters 7, 8 and 9 of this dissertation, [Alhagi08], and [Kumar07]) can be used to synthesize complete oracles or their parts for incomplete functions. Thus the approach of Parallel Quantum Computing can be also used as the machine learning method based on *learning oracles*.

15.3.3. To practically design oracles for Grover algorithm as quantum circuits the researcher has first to formulate various NP-complete problems and NP-hard problems as oracles or sets of oracles. Some robotic problems, especially in vision (such as convolution, matching, applications of Quantum Fourier Transform and other spectral transforms [Curtis04, Fan07, Breazeal02, Nielsen00, Perkowski07b, Waltz75, Wong89]) require quantum circuits that are not permutative but use truly quantum primitives like the controlled phase gate. The methods to convert these circuits to AQC model should be investigated and the problems should be converted to AQC model and executed on Orion. This material is beyond the present thesis because it is related to synthesis of non-permutative quantum gates, while this dissertation focuses on permutative circuits synthesis only. Hopefully, some methods developed here will be useful in the future

research of new students in the PSU quantum research group. Faisal Khan from our group is already working on a Ph.D dissertation on this topic.

15.3.4. Our group proposed an algorithm to find the best polarity Fixed-Polarity-Reed-Muller transform [Li06]. Several extensions to this method are presented in section 15.4 of this chapter. The presented general approach of representing unknown values as superposed values is very general and it can be used as another machine learning method when a function with don't cares (i.e. a set of "examples") is given at the inputs. Similarly the method presented in [Kumar07] is a general purpose machine learning method from examples which can be used in many robotics, Data Mining and learning applications.

15.3.5. In another approach, Quantum Neural Networks or Quantum Associative Memories can be used [Perkowski05]. There is already research at PSU on this topic by David Rosenbaum. In a non-published research the PSU group extended also the Quantum Fourier Transform based convolution/matching methods to Haar Transform, Complex Hadamard Transform and other spectral transforms [Perkowski07b]. Several image processing algorithms can be created for quantum computers with significant complexity reduction [Beach03, Curtis04]. These algorithms use not only the constraint satisfaction, SAT and search subroutines but also quantum spectral transforms and solving general purpose

Schroedinger equations. It is an open problem how they can be transformed to specifications for the Quantum Adiabatic computer.

15.3.6. In Chapters 12, 13 and 14 of this thesis, I developed oracles for classical problems such as SAT, maximum clique, exact ESOP minimization, maximum independent set, general constraint satisfaction problems such as cryptographic puzzles, and other unate/binate/even-odd covering problems, non-Boolean SAT solvers and equation-solvers. For all these problems we built oracles: in principle all these oracles can be converted to the AQC model of DWAVE. However in practice the Hamiltonians are so complex that the software should be developed to do this. It should be pointed that all our problems for oracles in Chapters 12, 13 and 14, although have simple formulations, are either used in practical applications or are very similar to more complex problems of this type that are used in practical applications. For instance, the logic puzzles are simplifications of certain logistics problems that have important applications in military operations and transportation planning.

15.3.7. The development of new quantum algorithms based on extensions and adaptations of Grover Algorithm, Hogg Algorithm and other quantum search and Quantum Computational Intelligence models is perhaps also possible. Generalizations of Grover, Simon and Fourier transforms to multiple-valued quantum logic [Fan07, Khan05a, Khan05b, Perkowski05]

as implemented in the circuit model of quantum computing should be considered. Analysis and comparison with binary quantum algorithms and their circuits should be performed. Methods of converting these problems to the AQC model should be investigated. This work is beyond the scope of this thesis, but I believe many quantum logic blocks and methods developed in this dissertation but specifically in Chapters 11 - 14 will be of extended use.

15.3.8. Generalizing the well-known quantum algorithms to multiple-valued quantum logic. For instance, in paper [Fan07] Yale Fan from our group generalized the historically famous algorithm by Deutsch and Jozsa to arbitrary radix and he proved that affine functions can be distinguished by this algorithm in a single measurement. Moreover, functions that can be described as "affine with noise" can be also distinguished. This can be used for very fast texture recognition in robot vision. Work on generalization of Grover to multiple-valued quantum circuits is also possible and will find applications in quantum robotics. Affine functions in general have interesting applications beyond those presented in this thesis. Moreover, using Chrestenson transform properties Yale Fan generalized [Fan07] the Deutsch-Jozsa algorithm [Nielsen00] for other texture recognition problems in robot vision tasks. PSU Group uses also the Grover algorithm

[Nielsen00] for robot action planning [Dong05, Dong06], problem solving and vision [Beach03, Curtis04].

15.3.9. Many problems listed above are useful in robotics to solve various vision and pattern recognition path-planning, obstacle avoidance and motion generation problems. Many NP problems from robotics and vision can be found in literature [Garey79]. Observe that every NP-complete problem can be reduced to Grover algorithm by building the respective oracle, and the Grover algorithm with its oracle can be further reduced to the AQC model that can be run on Orion. Similarly the classes of quantum simulation algorithms will be run using future DWAVE architectures. Although the speedup of the Grover class of problems is only quadratic, it will be still a dramatic improvement over current computers. It is also well-known and was demonstrated in previous chapters, that if some heuristics are known for an NP problem, one of several extensions and generalizations to Grover can be used, which may provide better than quadratic speedup. This approach is problem-dependent. Since however all classical solvers of NP-Complete problems that are used now in industry are heuristic and are usually more useful than their exact versions, I believe that the same will be observed when quantum programming becomes more advanced. It is not known yet what will be the speedup of problems from the "quantum simulation" class – it is an area of active research now.

15.3.10. The ideas proposed here in the framework of "Quantum Robotics" are new. They are different from the "quantum robots" proposed by Benioff [Benioff98] where a robot operates in structured quantum mechanics environment rather than in standard mechanics environment. Similarly, the robots from [Dong05, Dong06] are limited to only one aspect of mobile robotics, while the robots from Martin Lukac [Lukac07] are limited to emotional learning behaviors. The PSU model of a quantum robot, which may use quantum sensors but operates on normal effectors in standard environment is a generalization of the model from [Dong05, Dong06] rather than the original model from [Benioff98]. The PSU model of a quantum robot applies quantum concepts to sensing, planning, learning, knowledge storing, general architecture and movement / behavior generation [Lukac07, Lukac07a, Perkowski07a]. It uses quantum mappings as in [Raghuvanshi07, Brawn05], quantum automata [Raghuvanshi07, Lukac07a], Deutsch-Jozsa-based texture recognition [Fan07], Grover-based image processing, emotional behaviors [Lukac07], quantum learning based on logic synthesis [Fan07] and other models [Kumar07, Perkowski05a, Lukac03, Lukac07], motion planning and spectral transforms as its special cases. It is however this thesis that discussed for the first time how Grover algorithm can be used in selected robotics

applications, in particular to robot learning, including the learning of symbolic formulas as a special case of learning (section 15.5).

The algorithms and oracles introduced so far in this dissertation use the so-called classical "circuit" model of quantum computing. There are however also other models which may be implemented soon by physicists who work on new quantum technologies, and that may be will be even more practically successful than the "classical" quantum circuit model. The adiabatic model is only one of these new models. Although now we can only simulate quantum circuits using standard simulators such as QUIDDPRO on a standard computer, soon it will be possible to use the commercial prototype quantum computer from DWAVE Corporation [DWAVE07] to test at least some of our algorithms on a model of adiabatic quantum computer.

Concluding, when coupled with the truly quantum computer [DWAVE07], the quantum robots based on Grover oracles introduced here would speed-up all NP problems quadratically. Using variants of Deutsch-Jozsa and Bernstein-Vazirani generalized to multiple-valued logic some vision tasks would be speeded up exponentially, thus allowing to solve in real-time certain problems that are several orders of magnitude more complex than those solved by the existing computers [Fan07, Perkowski07a].

15.4. Machine Learning Using Spectral Approach.

15.4.1. General remarks about Machine Learning

As shown in chapter 5, the quantum algorithm for searching unstructured databases invented by Grover finds a number (or a set of numbers) that satisfies a certain constraint expressed by an "oracle". Here we describe a generalization of Grover's algorithm that finds the simplest expression of a certain form among all possible expressions for all possible solutions. The innovation of this approach is in finding (learning) a symbolic specification of a problem. The work presented in this section is an extension of paper by Lin et al [Li06]. This paper has motivated my entire dissertation. Our particular transform type used here is the Fixed Polarity Reed-Muller transform for which the number of non-zero spectral coefficients should be below certain threshold value. Thus our approach finds the particular FPRM form (among all 2^n FPRM forms) that has the minimum number of terms. In contrast to [Li06] where the completely specified function was considered, I observed that it is relatively easy to extend the approach from [Li06] to the incompletely specified functions.

Using this trick, in a standard way, the logic synthesis approach from [Li06] is made applicable to Data Mining and Machine Learning. Moreover and most importantly, the used by me representation of "unknown value" as the superposition is very logical. It is applicable to all other synthesis methods and Data Mining/Machine Learning

methods that include incomplete sets of examples. Observe that in next applications the data are incomplete.

In the most fundamental terms, our design here is based on a generator of all formulas for the problem specified as follows:

1. Given is a set of positive and negative examples (positive examples are true minterms, negative examples are false minterms)
2. All other minterms are treated as don't cares (unknown, or not presented examples).
3. This generator is controlled by a binary word, each selection of bit values for the control word creates another formula candidate (i.e. another FPRM transform from the family of all polarity transforms).
4. The generator is a quantum circuit so that the controls and the formulas can be superposed.
5. The cost of the formula is calculated as the number of terms (spectral coefficients of FPRM) that are non-zero.
6. The Grover algorithm is run to find such controls that the formula is as simple as possible (i.e. has as many zero coefficients as possible). In other words, Grover is run to find such input polarity vector that the cost of the solution expression is smaller than some threshold value *NX* and the solution does not exist for value *NX - 1*.

15.4.2. Oracle for completely specified FPRM.

The Quantum Oracle for the entire "Grover Architecture for FPRM Minimization", called the "FPRM Oracle" is implemented as a quantum permutative circuit that contains a subcircuit (butterfly) that expresses all possible FPRM solutions of a given function. This approach illustrates how butterfly circuits for fast transforms, as known from the spectral theory, can be combined with quantum computing ideas as a part of a Grover Oracle.

The original quantum search algorithm of Grover finds a single solution. This solution is a binary vector that satisfies the quantum oracle F. A quantum oracle can be considered as a Boolean function F with a solution minterm m_i that satisfies F (i.e. $F(m_i)=1$). Finding a solution can be thus visualized as finding a single number (cell) with value "1" (a true minterm) in a Karnaugh Map of function F in which all other cells have values 0. Obviously, when one solves this problem in the classical world and no additional information is available, the classical *SAT* algorithms can be employed. These SAT algorithms have worst-case exponential complexity. When there are $M > 1$ solutions, in the quantum search case one of many variants of the Grover's Algorithm can be employed to find all solutions (*SAT-ALL*).

As we remember from chapter 3, a generalization of PPRM is called the Fixed Polarity Reed-Muller (FPRM) form where every variable is either negated or not consistently

in the same polarity in every term of the expression. Thus, FPRM $F = a'b'$ has the polarity number 0 ($a = 0, b = 0$) and the equivalent PPRM $F = 1 \oplus a \oplus b \oplus ab$ has the polarity number 3 ($a = 1, b = 1$). As illustrated in chapter 3, in binary form, each FPRM represents a two-level circuit consisting of a set of conjunctions of literals (AND operations) followed by a multi-input addition modulo-2 operation (EXOR operation).

Several heuristic methods have been formulated in the past for both ESOP minimization [Sasao93, Mishchenko01] and for FPRM minimization (chapter 7 and [Sasao96, Dreschler96]). Here, however, I present a fundamentally new approach to FPRM minimization (with incomplete data) that is based on quantum logic and the use of Grover's algorithm. This approach can be extended to several canonical EXOR forms [Sasao96] as well as to the non-canonical ESOPs; however, in this section only the FPRM case is discussed. It can be observed that the method is based on controlling stages of butterfly diagrams and thus similar approaches can be applied to any spectral transform that can be described by some kind of a butterfly diagram. We present the binary case here, but the ternary case [Cheng05] is very similar. We will use the general blocks developed in chapter 11 (Figure 15.4.2.1). The FPRM Processor and the "Ones Counter" (Cost Counter) are built as in Chapter 11. The "Inverse Cost Counter and Comparator" is just the mirror circuit of the "Cost Counter and Comparator" circuit so it can be easily created by reversing order of inverse gates (a

standard method of "mirrors" discussed in chapters 3 and 7). Similarly the Inverse FPRM processor is designed as a mirror circuit.

Let us discuss this complex oracle in more detail. The entire proposed oracle, part of the Grover Loop, for finding minimum FPRM is shown in Figure 15.5.2.1. There are four blocks in this oracle architecture: *the FPRM processor*, *the Cost Function and Comparator*, and the corresponding inverse blocks *Inverse FPRM processor*, *Inverse Cost Function and Comparator.*

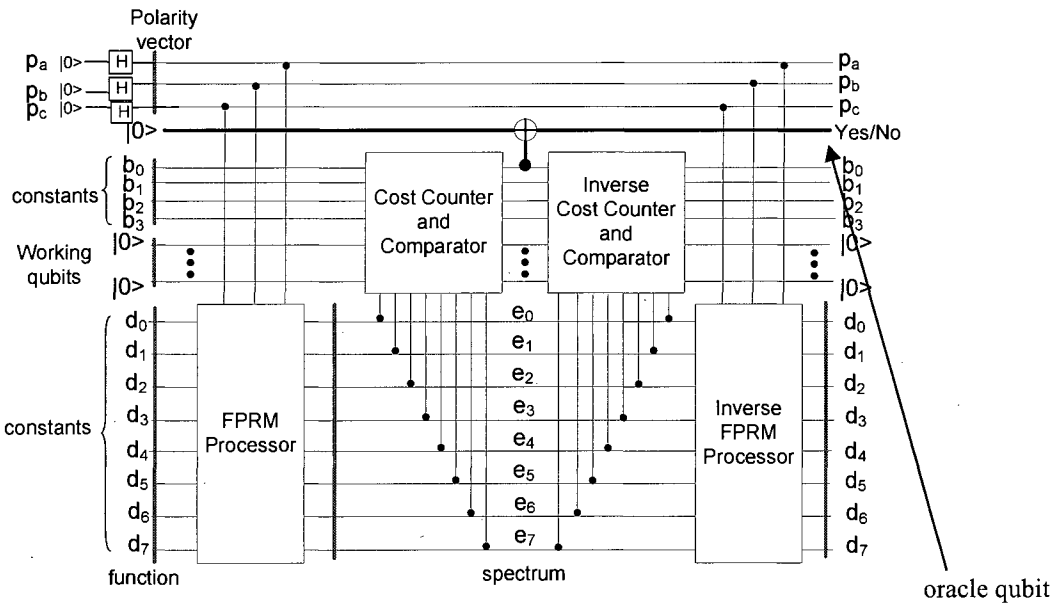


Figure 15.4.2.1: Quantum Architecture for FPRM Oracle for Grover's Algorithm. This is the case of 3-variable functions, there are thus eight minterms d_0 to d_7 and three qubits for polarity p_a, p_b and p_c (on top left).

Whether we solve the completely or incompletely specified function case, the inputs of the *FPRM processor* (see Figure 15.4.2.1) are:

(1) the binary values - the "vector of minterms" for a given Boolean function, and

(2) the polarity vector.

In case of complete functions only the care minterms, i.e. true minterms or false minterms are given as d_i. These values are constants: 0 for false minterm and 1 for a true minterm. Minterms are denoted as d_i, i = 0, …, 7 at the left of Figure 15.4.2.1.

The output of the FPRM processor is the binary vector of the FPRM spectrum coefficients for the given Boolean function and the polarity specified by binary (p_a, p_b, p_c). Observe however that this value is available only in quantum inside the Grover Loop. The measured output of the entire "Grover Architecture for FPRM Minimization" is only the polarity specified by binary (p_a, p_b, p_c). From these data a standard computer has to recalculate the vector of coefficients (e_0,.. e_7), but this can be done fast as no search is involved and the process is completely algorithmic as was discussed while presenting butterflies in chapter 3.

There are two input busses for the *Cost Function and Comparator* block, the threshold value and the polarity vector. The FPRM processor requires the 2^n sized truth vector of the Boolean function and produces binary values of the 2^n FPRM spectral coefficients corresponding to the function and polarity vector. Two tasks are

accomplished in the *Cost Function and Comparator* block. First, the number of ones in the vector of spectral coefficients is counted. Second, a comparison of the number of ones with the threshold value is accomplished. If the number of ones in the coefficients is less than the threshold value, the *Cost Function and Comparator* block will output a one, otherwise zero. The corresponding inverse blocks, *Inverse FPRM processor*, *Inverse Cost Function and Comparator,* accomplish the inverses (mirrors) of these functions.

The FPRM processor accepts:

1) a vector corresponding to the Boolean function and a
2) polarity vector

FPRM Processor outputs the FPRM spectral coefficients.

The core part of the FPRM processor is the "butterfly" quantum circuit. The polarity of the "butterfly" is controlled by the polarity bits. Figure 11.13.1 in chapter 11 shows the 1-variable FPRM processor which has a 2-bit function input ($[d_1, d_2]$) and a 1-bit polarity input (p). If $p = 0$, the output is positive polarity coefficients, otherwise, it is negative polarity coefficients. To understand this oracle in detail the careful reader should analyze the constructions of all blocks used in this oracle, as they are explained in chapter 11.

15.4.3. Oracle for incompletely specified FPRM.

In the case of incompletely specified datac the oracle is exactly the same as in Figure 15.4.2.1 but it is differently controlled. The qubits d_0 to d_7 are now set not only to Boolean values zero or one (for negative and positive minterms, respectively). These qubits must now correspond also to don't cares. The don't cares are created as shown in Figure 15.4.3.1 by using Hadamard gates. Every input qubit that corresponds to a minterm being a don't care goes through the individual Hadamard gate. Qubit d_7 in the Figure can be an example.

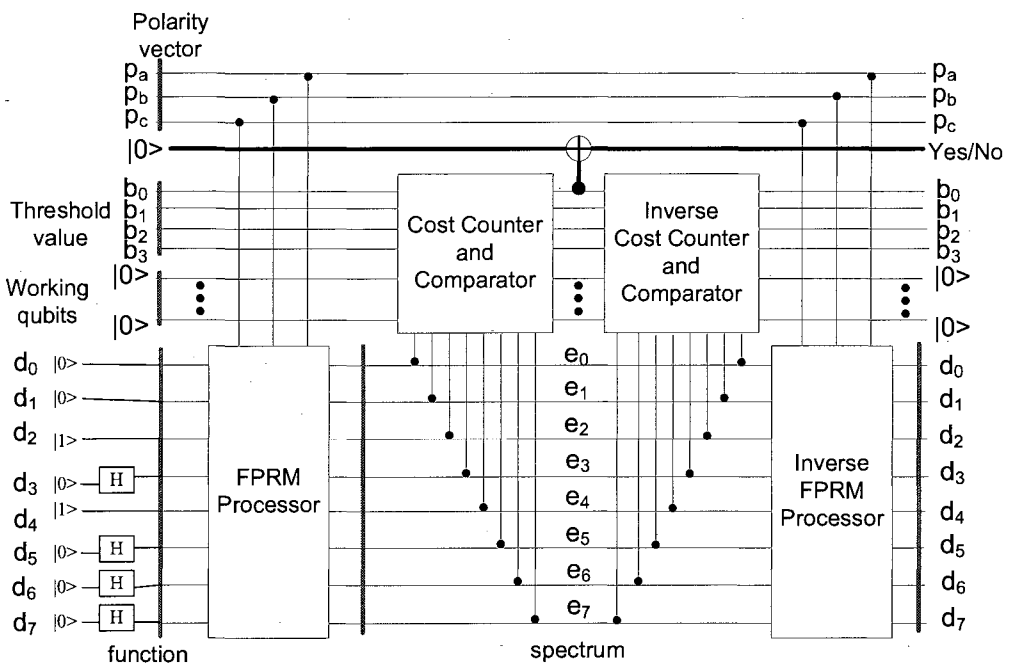


Figure 15.4.3.1: Quantum Architecture for FPRM Oracle for Grover's Algorithm. This is the case of incompletely specified function so all don't cares go through Hadamard gates. The circuit illustrates the incomplete function for which $d_0 = 0$, $d_1 = 0$, $d_2 = 1$, $d_3 = -$, $d_4 = 1$, $d_5 = -$, $d_6 = -$, $d_7 = -$.

The architecture from Figure 15.4.3.1 finds the cheapest solution for a given polarity p_a, p_b, p_c when the input of polarity qubits is fixed to a binary vector.

However, when the polarity is not assumed, which means the inputs p_a, p_b, p_c are provided through Hadamards (as in Figure 15.4.2.1), and the minterm data qubits are still as presented above, then the entire quantum architecture finds both: the best FPRM polarity and the best assignment of values to data minterms.

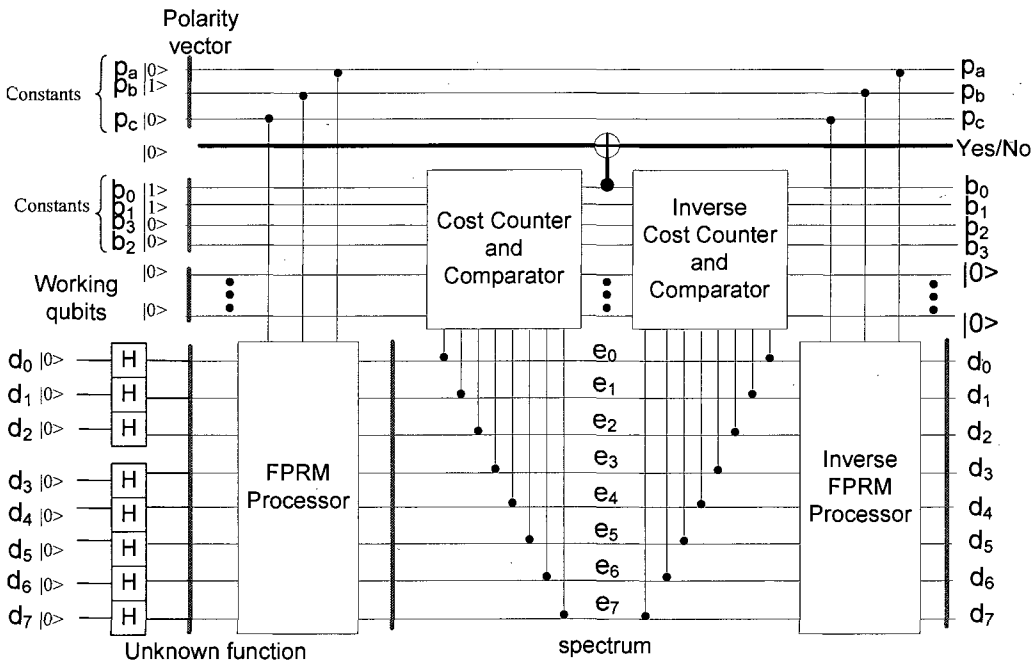


Figure 15.4.3.2: Quantum Architectures for spectral-based Oracle for Grover's Algorithm for the problem 15.4.3.2 from section 15.4.3.

Concluding on variants of search problems solvable with FPRM oracles, let us observe that we can formulate four different search problems. All these problems use exactly

the same oracle. Each of these four cases is selected by some specific way of providing input values to all inputs of the Grover's Oracle.

Problem 15.4.3.1.

1. Given are:

   a) a completely specified function *f* specified by the vector of its minterms.

   b) the integer number bound *B* on the cost *C(S(f))* of the solution *S(f)* being a binary vector.

2. The cost *C(S(f))* is the number of non-zero spectral coefficients in vector *S(f)*.

3. Find the FPRM polarity *Pi* for which the cost of spectrum *S(f)* is below the value of the bound B.

This problem is illustrated in Figure 15.4.2.1. It was discussed in section 15.4.2 and in the original paper by Lin et al [Li06].

Problem 15.4.3.2.

1. Given are:

   1) The polarity *Pi* as a binary vector

   2) The integer number bound *B* on the cost *C(S(f))* of the solution *S(f)* being a binary vector.

   3) The cost *C(S(f))* is the number of non-zero spectral coefficients in vector *S(f)*.

2. Find the function *f* such that the FPRM in this polarity *Pi* has the cost *C(S(f))* of spectrum *S(f)* that is below the bound value B.

This approach can be used to automatically invent new gates with small costs. This case is illustrated in Figure 15.4.3.2.

Problem 15.4.3.3.

1. Given are:

   c) The polarity *Pi* as a binary vector

   d) The function *f* specified by the vector of its minterms.

2. The cost is the number of non-zero spectral coefficients.

3. Find the bound such that this function *f* in this FPRM polarity *Pi* has the cost of the spectrum vector that is below the bound that is found.

This problem makes no particular practical sense but is added here for completeness and to show our general methodology of asking different questions to an oracle.

Problem 15.4.3.4.

Given are:

1. an incompletely specified function *f* specified by its care and don't care minterms.

2. The cost is the number of non-zero spectral coefficients.

3. Find the FPRM polarity P_i for which the cost of the spectrum vector is the minimum. Find this binary spectrum vector and the assignments of cares to don't care minterms.

This is the most important and useful generalization that we found. It is illustrated in Figure 15.4.3.1 with additional Hadamard gates on bits p_a, p_b, p_c.

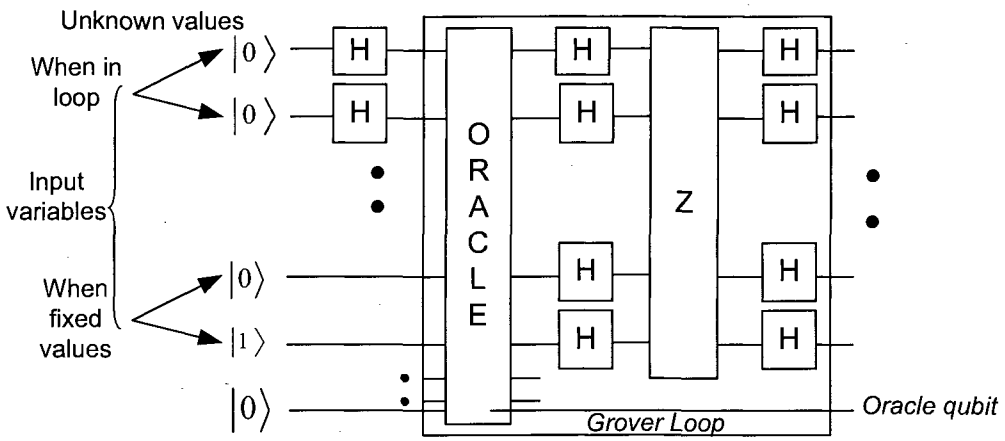


Figure 15.4.3.3: Explanation of using inputs for known and unknown values on inputs to extended Grover Algorithm. The known values are initialized to basis states $|0\rangle$ *or* $|1\rangle$ *. The unknown values are initialized to state* $\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$.

Concluding, the same oracle can be used to answer several questions, depending which data to it are fixed and which are unknown. Figure 15.4.3.3 shows the general way to create preprocessing circuits to oracles in Grover algorithm. Every input, for every subset of inputs with different meanings can be set to value $|0\rangle$ if this data bit is

negative, to $|1\rangle$ if this data bit is positive and to superposed value $\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle=[H]\otimes|0\rangle$ representing an unknown value that its definite value is searched for. This principle of creating oracles and formulating data for them is very general and I applied it to other problems not discussed in the thesis.

This principle can be summarized as:

"Create such an oracle that

(1) the care data inputs are fixed to binary values

(2) the don't care data inputs are set to $\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle=[H]\otimes|0\rangle$

and next measure all don't care qubits."

15.4.4. Generalizations and Applications of Spectral Learning Model.

15.4.4.1. Generalizations and applications of methods from sections 15.4.2 and 15.4.3.

Let us observe that the selection of FPRM spectrum as the spectral transform in the section 15.4.3 was purely incidental. As we know the FPRM is just one family of AND/EXOR spectral transforms. Because it has the simplest butterflies, the FPRM forms are practically the most popular. Therefore the authors of [Li06] selected this

family of spectral transforms. The approach illustrated in section 15.4.3 can be however applied to any family of spectral transforms, especially the transforms related to AND/EXOR logic. It can be also used to other transforms for which butterflies can be built as permutative quantum circuits, see Chapter 11, section 12. Hadamard, Fourier, Cosine and other transforms are for instance possible. This is because one can build in principle the oracles like in sections 15.4.2 and 15.4.3 for every family of expansions controlled by certain parameters. I tried to build oracles for such AND/EXOR spectral transforms as GRM and GPMPRM [Zhang99]. I found it possible to build such oracles, but very complicated. However, I make here a point that this is possible in principle. We can thus formulate the following generalized problems, each problem below generalizes the concept of an oracle from the FPRM oracle to every imaginable polarity-based (parameter-based, parameterized) spectral transform oracle.

Problem 15.4.4.1.

Given are:

1. The function F: I → O defined as a mapping (Boolean, Multiple-Valued or hybrid)
2. The bound B on the cost of realization of this function
3. The function is realized as an expression based on selecting some subset of non-zero coefficients of some spectral transform ST of this function.

Find

the polarity of this expansion (or equivalently, the value of the parameter)

for which the cost of the spectrum (number of non-zero coefficients) is below the given bound B.

Problem 15.4.4.2.

Given are:

1. The function F: I → O defined as a mapping (Boolean, Multiple-Valued or hybrid)
2. The cost of realization as the number of non-zero spectral coefficients.
3. The bound B on the cost of realization of this function
4. The function is realized as an expression based on selecting some subset of non-zero coefficients of some spectral transform ST of this function.
5. The spectrum
6. The polarity Pi

Find

the function F such that the given instance of the family of transforms in this polarity P has the cost of the spectrum that is below the bound B.

Problem 15.4.4.3.

Given are:

1. The function F: I → O defined as a mapping (Boolean, Multiple-Valued or hybrid). The function is realized as an expression based on selecting some subset of non-zero coefficients of some spectral transform ST of this function.
2. The cost of realization as the number of non-zero spectral coefficients.
3. The bound B on the cost of realization of this function
4. The spectrum
5. The polarity Pi

Find

the bound B such that this function F in this family of spectra in this polarity P has the cost of spectrum that is below the bound B.

Problem 15.4.4.4.

Given are:

1. The incompletely specified function F: I → O defined as a mapping (Boolean, Multiple-Valued or hybrid). The function is realized as an expression based on selecting some subset of non-zero coefficients of some spectral transform ST of this function.

2. The cost of realization as the number of non-zero spectral coefficients.

3. The bound B on the cost of realization of this function

4. The spectrum

5. The polarity Pi

Find

the polarity of the expression within the family of transforms for which the cost of spectrum (the number of non-zero coefficients) is the minimum.

This is the most important and useful generalization of generalizations, as it relates to quantum, Grover-based machine learning with arbitrary spectral transforms.

Applications of methods from section 15.4.2 and 15.4.3 can be in all those areas of research and practical technology in which the spectral transforms are now being used. This includes the following:

1) General Logic design (also logic minimization for reversible and quantum circuits themselves),
2) Logic Design for Test, highly testable circuits,
3) Image processing and DSP,
4) Data compression,
5) Communication,
6) Cryptography,

7) Error detecting and error-correcting codes,

8) Machine Learning.

In section 15.4.2 and 15.4.3 only a particular approach was illustrated because as the spectrum transform we used the family of FPRMs, the reader remembers however from chapters 3, 7, 8, 9 and 11 that there are many parameterized AND/EXOR forms such as GRM, GKRM, KRM and also many other spectral transforms such as adding, arithmetic, generalized, Haar Transform, Fourier Transform, etc. Each of these families can be used to build the generator being the part of the oracle. We found the way to build generators for all these forms. By the way, each of these generators can be created similarly to the generators for FPRM in this chapter, based on the knowledge of the butterfly structures [Perkowski97a, Perkowski97b, Zeng95]. What is most important is that we found a general method to create quantum oracles for all problems described by families of spectral transforms. The Zhegalkin hierarchy finds therefore one more practical application. Such families include not only the classical AND/EXOR transforms but also the entire (some new, some old) families of Haar transforms and wavelets, polarized Walsh, complex Walsh, complex Haar, multivalued Haar and Walsh (Hadamard, Paley, Karczmarz), Chrestenson and Fourier. Some problems in signal processing and image processing use adaptive filters that change dynamically the transform applied in real-time as an approach to adopt to the changing environment. Our approach is applicable directly to all these problems with no any modification. This is a very general method with many new applications.

15.4.4.2. Applications in Quantum Game Theory.

It is difficult to predict future of technology but one can observe that general methods to represent data such as the spectral methods find applications in very many areas and are used in many commercial products, the Cosine Transform with applications in JPEG and MPEG can be just one convincing example. Slant and Haar transforms were used by Intel and other companies. Reed-Muller codes are used in interplanetary communications. The list goes on. In addition, there is also a quickly developing area of quantum game theory where many known results from quantum algorithms were assigned very interesting new interpretations. All problems represented above as the design, construction, mapping problems are interpreted as games between two or more participants. This will have applications in economy and Internet gambling with future quantum internet. For instance, the learning problem discussed above is more general than the known quantum game of finding the conjunctive formula of literals for a given set of data. Our machine from previous sections could be just set to the threshold of two (limit to a single product of literals) to obtain a product of literals (not necessarily a minterm like in many games) that satisfies the input data being formulated as a set of minterms. In quantum games the number of players is the number of qubits and the number of strategies per player is the radix of logic used in the respective qudit. Therefore, all circuits presented here can be generalized to ternary quantum gates [Khan05, Kalay99c], allowing to create ternary butterflies [Cheng05] and more efficient arithmetic for larger counters and comparators. Next

they can be generalized to arbitrary radix of logic and applied to respective games. The Classical Game Theory finds many applications in robotics, especially to military and social robotics. We can speculate that the same will happen to the Quantum Game Theory.

15.4.4.3. Advances in the design of quantum arithmetics.

There is one more aspect which should be discussed at this point. In this and previous chapters we showed how Grover's algorithm can be extended to practical problems in classical logic minimization and Machine Learning. Thus our examples illustrated also the design of practical reversible circuits using quantum gates for blocks that will be normally incorporated inside oracles. In many oracles that we tried, always exist arithmetic blocks such as adders, subtractors, comparators, counters of ones (compressors), butterfly transforms and logic blocks. In chapters 10 - 14 we showed usefulness of some of these blocks. I see that much more work can be done on arithmetic of quantum computers, but this is beyond this thesis. Dr. Mozammel Khan from Dhaka published several papers with Prof. Perkowski on these topics and several authors from Bangladesh were included. Hopefully this new research area will be expanded worldwide.

15.4.4.4. Quantum oracles for learning based on non-spectral approaches and types of transforms.

The next goal of research in the area of Machine Learning and Data Mining can be to design and simulate algorithms similar to those listed above that would use representations based on Galois Fields other than the GF(2) field [Kalay99c]. The goal of these algorithms would be to create ESOP-based and GFSOP-based optimized quantum arrays and quantum state machines. In terms of Machine Learning the above methods would be characterized as classical "Occam Razor" learning methods.

We look in them for a formula of certain type (like a GFSOP expression) that has a cost as low as possible and has either no error or an error smaller than some Error Threshold Value. The desired cost is the smallest possible number of non-zero spectral coefficients. The type of the circuit synthesized is the learning bias of the method.

To my knowledge this kind of learning algorithm was never proposed to be implemented in a quantum circuit. So far only learning of DNF formulas and Neural Nets was applied in the area of Quantum Computational Intelligence [Ventura98, Ventura99, Behrman96, Hopfield82, Perus96].

When one discusses the choice of the spectral transform in the oracle, special attention must be paid to Karhunen-Loeve transforms [Thornton05] because these transforms, although difficult to implement, allow for the best approximation for a wide set of base functions and are therefore the optimal spectral transforms. Wavelet transforms

should be also analyzed as a candidate for transforms used in quantum oracles. The PSU Group intends also to build quantum oracles for other spectral transforms, especially those used in robotics, and particularly in robot vision. One of the aims of the entire work of PSU group is to discover practical and efficient methods of designing binary, multi-valued and hybrid quantum circuits, blocks and algorithms not for random benchmarks but for practical blocks and architectures that have practical applications in quantum oracles of algorithms that speed-up very time consuming algorithms. The work presented in this thesis can contribute to this broad task by delivering several practical circuits, re-usable blocks (such as sorters or comparators), quantum algorithms (such as graph coloring) and partial methodologies. The proposed approach can be also applied to any problem (like filter design or processor design) described by:

1. certain discrete parameters (encoded to binary for our algorithms)
2. there exists a cost function based on the complexity of the output data (the number of the FPRM transform coefficients in case of sections 15.4.2 and 15.4.3)

We created thus, particularly in this chapter, a general quantum method to solve many classical problems in image and signal processing, filtering, matching and learning. In particular, every problem for which there exists a Linearly Independent Transform can be solved this way, which includes wavelets, Fourier transforms and other "orthogonal transforms".

CHAPTER 16

CONCLUSIONS

16.1. What can be found in this concluding chapter

Initially, the main goal of the dissertation was to develop quantum oracles for many interesting problems to be able to evaluate the Grover algorithm. Simulations using QUIDPRO and Matlab proved that small oracles that I designed using my new methods can be verified and that they are correct. I used also the new simulator from Professor Miller. This simulator allows simulating also multiple-valued quantum logic and larger circuits than QUIDPRO. With respect to technology I selected NMR as it is confirmed by many experiments and the one that is most well-known.

When we were sure that the concept of quantum oracle works well and we were able to simulate our oracles for Grover Algorithm, a new problem appeared "how to design the oracles" as my first oracles were not designed systematically. This design problem exists on many levels:

1. designing oracles on level of logic blocks,
2. conversion of non-reversible blocks to reversible,
3. synthesis of blocks (circuits),
4. synthesis of gates for circuits on level of their quantum realizable components,
5. synthesis of small gates and components on level of quantum pulses for a given realization technology.

I found that the existing CAD software even for reversible or quantum circuits is not specialized to design oracles. Therefore I decided to create software for exact solutions and software with visualization.

The previous chapters included some new ideas for future work that result from the ideas and methods of the thesis. There are also some other general ideas that came to my mind while writing the thesis, but were too general or too broad to be included in the thesis. Therefore in this chapter I will conclude and evaluate the main ideas of my thesis and I will discuss certain very general approaches that result from the research material presented in the thesis.

16.2. Evolutionary Darwinian algorithms versus Evolution of Quantum States

As discussed in the thesis, one of my synthesis methods was evolutionary computing. A question may arise that I permanently keep asking myself: "is GA a good approach to the design of quantum circuits?" Let us first observe that natural systems are extremely well adapted to their environments. In a sense, the systems result from their environments. It is found in Nature that the structure of all organisms has been designed to provide the capability of solving a multitude of complex problems for both survival and growth, through instinctual, experiential, and intellectual means. Over the millennia, it is these capabilities that have proven an effective method of sustaining

the existence and the propagation of natural organisms. Even as environmental conditions change, it is the continuous adaptation process of natural organisms adjusting to their environment, through evolutionary processes, which sustains life. As Nature itself has proven its development of robust system design, we decided to use the biologically inspired evolutionary processes in the thesis, as a part of our approach to design quantum circuits. In the process of my learning and experimenting, I found that other approaches are possible, but GA was a good starting point. It lead also to some philosophical questions.

It remains an interesting and fundamental question if the quantum-mechanical search, which is used by Nature to solve physics problems, is also used by Nature in the living organisms for optimization and "problem-solving". Positive answer would be a great discovery but would be not surprising in the light of a common belief that Nature uses "Genetic Algorithm" to improve species [Drechsler93, Dill97, DeGaris92, Goldberg98, Holland92, Higuchi97, Higuchi97a]. We formulate in the thesis the problem *"what is the relation between the evolution of quantum systems and evolution of living organisms"* but we do not attempt to answer this difficult question in this dissertation. This is left for future work. Let us only observe that some recent published research treats evolution of the early Universe as an evolution of a quantum computer.

The Genetic Algorithm technique provides a means for applying the evolutionary process within an artificial system, in our case, within a quantum circuit of certain type. The Genetic Algorithm is a process that evolves problem parameters directly or through the evolutionary process of natural selection. An artificial evolution is applied to (software) data structures, consisting of functions (mathematical operations) and terminals (variables), to develop algorithms (rather than particular solutions) capable of problem solving. Through a process of emergent intelligence, the GA and its Quantum counterpart evolutionary algorithms formulate engineering solutions based on an accumulated knowledge of the problem and the merit of potential solutions. In my opinion, based on my experiences gained while writing the thesis, the designer should be not dogmatic about using the GA algorithm "taken from books" but rather he should treat evolutionary ideas as well as quantum ideas as powerful "computing metaphors" to create his own problem solving and learning approaches. This was an approach presented in this dissertation and it lead to solutions better than using only a purely GA-based method. Further work of combining algorithmic and (GA) search methods should be continued with applications not only to quantum circuits but also to quantum automata and quantum algorithms.

A question may arise what is new in my approach to using GA in quantum circuits design. It is well-known that in recent years, the Genetic Algorithms, as machine learning techniques, have been successfully applied to a wide range of engineering

problems as diverse as graph coloring [Lewandowski94], traveling salesman [Kruska56], quantum circuits design [Perkowski04, Lukac02, Lukac02a], Neural Network design, economic trend prediction, control theory, and firmware development, to list just very few. They have been successfully used also to quantum circuits design [Giesecke06, Khan03, Lukac02, Lukac02a, Lukac05]. In my opinion a new asset of this dissertation was to combine the GA with other search types. Another new aspect is the concept of parallel quantum search that uses GA as one of its methods.

Another question that people asked me is this: "with the possibility of realization of quantum algorithms in all kinds of reconfigurable quantum computer hardware, why you think that the Darwinian evolution principles should be used in the quantum world". One can think that may-be quantum evolution mechanisms are sufficient and that the evolutionary and neural quantum concepts become superfluous. This idea was suggested recently by the inventor of Evolvable Hardware, Dr. Hugo De Garis. He believes that quantum search is sufficient and GA will be useless in quantum computers, despite he wrote himself many papers about quantum genetic search. There were also no good examples of using evolutionary methods on practical quantum circuit synthesis problems. Such practical illustrations are entirely missing from the literature on the subject [DeGaris92, Coon94, Drechsler97, Drechsler99, Disman96]. (This remark does not concern the research of our group, but was a starting point of

my work in 2004). Concluding, it is not certain what will be the role of genetic algorithms in quantum area, although several authors work recently on "Quantum Genetic Algorithms".

In any case, regardless of future works of humans, Nature used already both "quantum evolution" and "Darwinian evolution". The Darwinian evolution was "invented" very late by Nature, so it is perhaps subsumed by some more general types of evolution. The Universe was able to evolve much before the first living species arrived. So, if we believe in Darwinian Evolution, how the Darwinian Evolution evolved from some earlier evolution which could be only chemical and physical? Did Darwinian Evolution exist before the first living cell arrived? Can something be created from nothing? Therefore I believe that Darwinian Evolution results from Quantum Mechanical Evolution and may be other evolutions (chemical, physical) that were created in between.

Virtually no research papers to date have applied any of these "combined evolutions" concepts to logic design and optimization, as well as to the closely related machine learning methods based on logic synthesis approaches [Dill01, Lukac08]. We also did not discuss these issues in the dissertation. The future research should include relations between quantum and Darwinian evolutions and my "engineering research" should use these results to the synthesis of quantum circuits and algorithms.

16.3. Links of our methods to Machine Learning and Data Mining

In current and forthcoming "Age of information", Data Mining is and will become even more so a necessary and ever increasingly important technological tool with very wide applications areas. These techniques will be for example built into home robots that will predict behaviors of children, elderly and non-sophisticated users to communicate safely and fully with them by using natural language and human-like gestures. Development of future Data Mining methods will require very powerful computers, orders of magnitude more powerful than any computers currently in existence. In general, as the current trends in both business and machine learning databases demonstrate increased size and complexity, it is even more critical to delineate useful information, or knowledge, from raw data (like camera, microphone, chemical sensors and quantum sensors in future) by automatic means. The recognition, discovery, and analysis of rules or patterns within real-life-data-created databases and information streams of extremely high bandwidths will become critical to all these processes. Again, nothing has been published so far about data mining using quantum computers, although it is obvious that Data Mining of extremely large automatically created data bases will become a very powerful application of future quantum computers.

This thesis is the first one to propose these ideas (but with no details). Here we speculate that:

1. Quantum computers will collect large amounts of small pieces of information that are now not possible to be analyzed in current technologies by standard computers (or humans). These types of data would require the complexity of analysis that is currently impossible, in order to create extremely complex models. These complex models will allow to better explain some phenomena or processes (like those in quantum mechanics, stock market, or weather prediction).
2. Quantum computer can analyze certain minimally different statistical results like the influence of heat on a given roulette wheel in a particular casino and thus, it will be able to make the real-time predictions which will be superior to any human or classical computer. Modern computers have no ways to acquire or process such extremely high volumes of data. And they never will as the capacities of quantum associative memories are exponentially larger than those of standard computers.
3. Quantum Data Mining will be thus a fruitful research area in coming years, together with Quantum Game Theory [Khan05, Khan06, Eisert99, Meyer99] and Quantum Markets [Pakula06]. It is expected that they will be much applied to study market behaviors and predict moves of competing companies.
4. Those societies without quantum computers will loose the battles in economy and in real war battlefields, as *"our" soldier robots with quantum brains will*

outsmart "their" soldier robots with classical computer brains [Perkowski's slides for Intelligent Robotics class Fall 2007].

The requirements of Data Mining (DM) and Knowledge Discovery in Databases (KDD) can be compared to those of the traditional logic synthesis. In common, both applications share the goal of fully describing the system with a minimal rule set (or formula), as required by the Occam's Razor Principle [Gamberger97]: "If two theories explain the set of facts equally well, the simpler theory is better". However, the system specifications in the DM/KDD field and the logic synthesis field differ much. While the DM/KDD approaches seek to discover new patterns and rules from the data, the logic synthesis describes circuits minimized by the number of gates, levels, and literals utilized. Finally, *"the biggest difference is that most circuit-related multi-valued logic problems are nearly completely specified, while functions in machine learning tend to be 99.9% unspecified in their respective learning domains"* [Files97]. A number of methods of data analysis have been demonstrated in recent years, including Neural Networks [Hagan96], decision tree generators such as C4.5 [Quinlan93], Function Decomposition [Files97, Files98, Files98a, Burns98] and others [Berry97, Alexits61, Perkowski95a, Kosko94]. While these methods are effective for machine learning, they are not readily applicable to both circuit design and Data Mining. This is apparent since the requirements for quantum circuit design are much more stringent than that of machine learning; an implementation must be constructed so that the logic

function is always 100% correct (although it may be not true for some applications of oracles where quantum computer only proposes a solution with high probability of success but standard computer is still necessary to verify precisely the correctness of what was guessed by the quantum accelerator). This degree of certainty is not necessary for machine learning applications. Recently proposed Quantum Computational Intelligence techniques such as Quantum Neural Networks and Quantum Decision Trees [Farhi98] obtain in theory good results, but cannot guarantee convergence to error-free solutions. There are also other quantum Computational Intelligence approaches proposed, for instance some use other network types or concepts. All these approaches together suggest certain future convergence of the research areas of Machine Learning, Quantum Mechanics and Logic Design/Evolvable Hardware. I believe that much work will appear soon in the area of Quantum Computational Intelligence and that the ideas introduced in this thesis may become a starting point to at least some of them.

16.4. Links of our methods to Evolvable Hardware. Towards Quantum FPGA

Recent interest in the field of Evolvable Hardware (EHW) [DeGaris93, Hemmi94, Higuchi93, Higuchi94, Higuchi97, Higuchi97a, Thompson95, Thompson95a, Thompson03, Sipper97] has been demonstrated in the area of Computational Intelligence, in the FPGA IC design community, in the FPGA user community and in

the robotics research community. The EHW field has emerged as an outgrowth of the development of computational and artificial intelligence learning techniques, as well as advances in Artificial Life theories. The ultimate goal of the evolvable hardware research is to automatically produce highly complex electronic hardware circuits that can architecturally adapt (either "on-line" or "off-line") to environmental variables, as deemed necessary. The promises of this new technology, once mature, are that evolvable hardware will dramatically reduce the traditional engineering development time and further, that the developed circuits will be highly fault tolerant, quick to implement (eliminate device reprogramming and re-design time), operate at higher speeds than software learning technologies, and produce highly advanced architectures. It is believed that these architectures will be quite different from the traditional design synthesis, optimization, and partitioning schemes (a combination of synchronous and asynchronous designs, different types of "module" divisions, etc.). When the EHW theory is fully developed, this new technology should allow a completely automated creation of highly complex circuits. Circuits requiring thousands of man-hours of development time, or perhaps, circuits even more sophisticated in design will be created with no human intervention at all. These circuits will be self-learning, highly robust, gracefully degradable, and can be modified as needed. It has been hypothesized [DeGaris92, Buller03] that in the future, as electronic circuits become increasingly complex, and will perhaps consist of billions

of nodes and internal interconnections, the evolvable hardware approach may be the only design tool possible for practical development.

The implications for such a powerful technology, which is capable of self-directed learning and adaptation, will also present some interesting philosophical questions for our understanding of life and intelligence. The genetic evolution of machine learning, knowledge discovery/data mining, and circuit design is the first step in the development of this powerful scientific trend and industrial technology towards quantum technologies of the future [Perkowski99c, Negotevic02]. Again, very little, if anything, has been published about quantum evolvable (reconfigurable, adaptable, learnable) hardware which is proposed in this thesis for the first time (although similar concepts appeared in the literature in the course of writing this thesis (successors to [Nielsen97]). Observe that evolvable hardware currently exists in the domain of Field Programmable Gate Arrays (and their analog equivalents called FPAAs – Field Programmable Analog Arrays). On the other hand, it is recently believed that quantum computing will be more similar to building accelerating co-processors for standard computers and realized in something like "quantum FPGAs", rather than similar to mainframe microprocessors. Observe that this is exactly the approach that has been developed and illustrated in my dissertation.

It is obvious that the concept of quantum evolvable hardware that could theoretically find exact minimal solutions to problems that classical computer would be never able to solve is not possible with traditional software GA approaches. The quantum speed-up is in the sense "given for free" in Grover algorithms once the correct setup is achieved. Therefore future evolvable hardware should use both Darwinian and Quantum Evolutions, where the Grover algorithm is only one example of a quantum evolution and the "Quantum GA" of Hugo De Garis is the only one example of combining quantum and biological evolutions.

All these considerations led us to the model of this thesis in which the future quantum computer will be a parallel system of hardware-programmable units, both classical and quantum, while quantum units will use Grover algorithm. This model is different from De Garis model, Han/Kim model, or any existing model of quantum computing. Let us stress again that when compared to the two existing standard computing technologies, a general-purpose processor and FPGA, the quantum computer is much more similar to the last one. This powerful similarity was used in this dissertation to develop new methods. These methods were innovative especially in light that even in standard computing the research community is still not sure about all capabilities of adaptable reconfigurable massively parallel architectures based on multiple FPGAs. Much further research in this area is possible, and this thesis is only the "beginning of beginning" to investigate combining quantum evolutionary and other methods.

16.5. Our approaches do not belong to the family of "quantum inspired algorithms"

Another comment seems appropriate at this place. The so-called "quantum-inspired" algorithms originating from KAIST in Korea (Dr. H. Han and Professor J. H. Kim) are performing better than the classical GA in some scheduling and other practical problems [Han00]. One can consider using these algorithms for our applications. However, let us make a point that I did not work on this topic in my dissertation. First, because we believe that better methods can be found, second that I wanted to develop my own method, which in future may be compared with the approach of the Korean team from KAIST.

The "quantum-inspired" algorithms still belong to the classical evolutionary paradigm, with chromosomes, crossovers and mutations, etc. This is contrast to other Quantum GA algorithms recently published which are more similar to our approach. However, certainly the comparison of my work to the works of Han and Kim would be useful in the future.

The issue of the interrelation of Darwinian and Quantum Evolutions in many of its possible aspects remains a fascinating research topic. For what reason should anybody believe that the Darwinian evolution, as based on chromosomes, crossovers and mutations and simulated in GA software, is the only evolutionary process created by

Nature? The answer is not known. The software simulations so far prove that the quantum evolution can play the same role as mutation and crossover in some software applications. The question is then this "may be quantum evolution contributes also to species evolution in our real world? " Thus, quantum evolution of not only the inanimate Universe but also the living Universe?

May be there is also a quantum component in the evolution of animals? Some new theories speculate that quantum mechanics is a part of neural processes in humans (works of Penrose and Hameroff [Hameroff96, Hameroff98] and [Lukac PhD thesis]). We believe that similarly, quantum components will be added in future to immune system modeling, fuzzy systems and evolutionary/reinforcement mechanisms that are used now in "Computational Intelligence" research area to describe decision and optimization processes. My belief is based on the fact that it is just the quantum physics (or even some more complex physics like the string theory physics), and the logic based on this physics, that really exist in Nature. The classical logic and classical (Newtonian) physics are just early human approximations which simplify models for user convenience or because of creators' ignorance. They do not represent the logic of Nature. An imprecise model may be thus a Neural Network or Darwinian Evolution, while the precise model must include quantum nature of everything. (Just one citation from Hameroff about quantumness of microtubules: *"Microtubule based cilia in rods and cones directly detect visual photons and connect with retinal glial cell*

microtubule via gap junctions". http://www.quantumconsciousness.org/penrose-hameroff/quantumcomputation.html). Researchers like Roger Penrose and Stuart Hameroff speculate that all intelligent behavior uses the quantum mechanics as its physical aspect, rather than using the classical physics only. They believe that quantum mechanics is responsible for our conscience. Regardless whether the human brain is quantum or not, future robots will have quantum brains and future computers will use quantum mechanics. It will be so, simply because only a quantum computer will allow solving problems of many orders of magnitude higher complexity that will be necessary for these robots to operate and survive.

Observe also that the quantum logic is different from the classical logic (the classical logic is the logic of Aristotle formalized by Boole) on which modern computers and all software modeling human reasoning are built. Several attempts to create more useful logic like multiple-valued logic of Lukasiewicz and Post or fuzzy logic by Zadeh [Zadeh83, Zadeh96] proved useful in some applications. But it is only now, with the arrival of quantum computers, that we are learning and trying to understand the logic that is used by Nature. When we will learn the "logic of Nature", we will be able to build and program totally different, more Nature-like, computers. This is the main principle of the dominating recently research direction known under several names: "Physics is Computing", "Universe is a computer", "Natural computing", "Nature-inspired computing" and "Building computers modeled after Nature". Thus, we

believe that it is interesting to look for a new approach to quantum logic rather than just apply known evolutionary ideas to the quantum domain [Lukac02, Han00] (their approach by the way is the research standard now). Again, these new approaches can be hopefully built on top of some ideas of my thesis.

16.6. Are our search models from this thesis realistic

Coming back to our way of modeling quantum phenomena in CAD computers, some of our methods like the FPRM minimization assumed that the computer has at least as many quantum wires (qubits or qudits) as there are all minterms of the function (their number is exponential). This is a huge number, so even when quantum computers will become available and practical for other problems like secure communication and cryptography, only small CAD problems will be solvable with the approach proposed in my dissertation. This may be a valid criticism of my work, but first, this criticism can be applied to many other published quantum algorithms, and second with time progression even larger quantum computers will be built. It is now difficult to predict how many qubits will exist in a quantum computer 200 years from now. Like nobody would predict in year 1850 the power of standard microprocessors of year 2005. In nineteen century, when Babbage and Boole speculated on power of "future computers" and all the London's elite treated Lady Lovelace, the first computer programmer ever, as an eccentric if not crazy person because of her opinions about the possible power of

computer programs, how can the philosophers know that it was Babbage and Boole and Lovelace who were right!

Therefore, this whole thesis is based on an optimistic speculation of existence of very powerful quantum computers (i.e. with very many qubits). We accept therefore a potential criticism that this dissertation relates to the kind-of "science fiction" technology, but we observe that all quantum computing research other than building small prototypes was science fiction in this sense in year 2005, when I started to work on this thesis. There was a dramatic flurry of new fundamental developments in quantum computing in years 2006, 2007, 2008 [Wikipedia]. I believe therefore that this research is very important because it analyzes which problems that are unsolvable now, will become potentially solvable with the arrival of more powerful quantum computers.

16.7. The main idea of quantum search in this dissertation.

Let us now try to conclude in few words the very basic, central idea of this dissertation and its potential continuation. The basic principle of our approach is very simple but therefore extremely general. First, the logic function representing solutions to some problem (an oracle or its part to be synthesized) is specified using a truth table. All true and false minterms are encoded as quantum wire states initialized to basis quantum states $|1\rangle$ and $|0\rangle$, respectively. All don't care minterms are created using the

well-known quantum Hadamard gates as all possible superpositions (details explained in chapter 2). Every "for all" loop of the algorithm goes through all "don't cares" calculated as full superpositions. This way the quantum algorithm combines constraints (the cares) with free choice (the don't cares) using quantum evolution controlled by the (configurable, quantum) processor according to the design construction and the parametrical choices of the oracle/algorithm designer. The fundamental difficulty of standard logic synthesis algorithm for don't cares is solved automatically in this approach from my dissertation. It is a byproduct of the superposition principle of quantum mechanics, i.e. our method uses Hadamard gates that create the equal superposition of all possible basis quantum states - the synthesized logic expression is derived for the cares and don't cares of the specification. This solution expression results from the quantum evolutionary process in which superposed states are propagated, and next the quantum amplitudes are amplified and measured (as in the Grover algorithm and its variants). Observe that treating don't cares in classical logic synthesis algorithm was always very difficult. In quantum it is easy, because a don't care is just a "logic one" in one quantum Universe and "logic zero" in another quantum Universe, so the exhaustive nature of quantum superposition itself (all binary combinations after parallel Hadamard gates – [Nielsen97]) solves the problem automatically. This is a very powerful general principle of formulating and solving optimization problems with incomplete

information using quantum circuits. This approach can be used to any constraint satisfaction problem.

It is this broad application of the Quantum Search to Logic Synthesis/Minimization and automation of the design process, which differentiates our research philosophy from all other circuit design methods known so far (with the sole exception of the Lin/Thornton/Perkowski paper, the source of my inspiration in this thesis).

This outlined above quantum hardware search/design technique is also a multi-purpose design approach, offering great flexibility. In contrast to other approaches to logic synthesis, this method of circuit design can be completely customized to optimize for virtually any cost function, i.e. circuit area, power, delay, number of gates, number of inputs, circuit speed, etc. Everything depends on how the part of oracle that calculates the cost function is constructed from quantum gates, circuits and blocks. Therefore, after our explanations and illustrations in previous chapters, everybody who understands classical logic design and disposes reversible synthesis CAD tools can design such circuits. It is the designer who can take into account any conditions, constraints or parameters by building a respective oracle. *The synthesis problem becomes the oracle design problem.* This requires the ability to design large hierarchical oracles efficiently. In far future, the desired optimization goals and their relative importance will need only be described to the Quantum Search Problem

Solver by a numerical judgment of the proposed solution's merit. Now the designer has to design in full detail all the specific quantum circuits (as it was explained in sufficient details in chapters 10, 11, 12, 13, 14 and 15).

16.8. Brute force Search versus human-like intelligence

Another comment seems appropriate in this conclusion as it also relates to the general philosophy of my dissertation. At the early phase of building chess programs it was believed that mimicking the thinking processes of the chess grandmasters is the "way to go". It was however shown in 50 years of chess computer research that a massive parallelism with no "human" intelligence is a better approach. IBM just needed a massively parallel classical hardware Deep Blue with exhaustive search implemented in it to win with the world chess champion Kasparov. The non-informed ("stupid") search is what the quantum computer can do better than any existing computer, as the Grover algorithm can speed-up this search from O(N) to O($\sqrt{N}$) [Grover96]. On some problems the speedup can be even exponential, but the science knows so far only very few such problems. Thus, may be many problems will be solved in future just be the sole power of exhaustive search made possible through quantum parallelism.

Is however quadratic speedup enough? Of course all quantum researchers have ambitions to find new algorithms with exponential speedup, but even quadratic speedup is a revelation.

One may say "Grover algorithm has not a great speed-up", but critics forget that this speedup is only when we want to have 100% probability of success. Much smaller number of steps may be sufficient to generate a good guess that is next verified by a standard computer. For instance, Professor Julian Miller shows recently that a constant complexity O(1) is obtained for quantum SAT problem with many solutions [Miller94a, Miller94b]. Remember that for NP problems we can always verify any guess made by a quantum computer with the classical computer quickly, and we do not need the exact minimum solutions in most cases. (It is well-known that for all NP-complete problems the solution correctness verifying is much easier than finding the solution). Moreover, quadratic speedup is sufficient in many areas such as when compiling a software program or playing robot soccer—there is a big practical difference between $2^3 = 8$ seconds and $2^6 \sim 1$ min. For example, this quadratic speedup becomes extremely useful and efficient when the algorithm works for longer period of time such as that when a classical algorithm takes 10 hours, it will take for Quantum Grover Search ~ 24 minutes. If for classical algorithm 100 hours, for Quantum Search it may take ~ 77 minutes only. So, we can easily see the difference and appreciate the tremendous speed up of Quantum Computing (the above example was for Grover Algorithm case only).

16.9. Exact versus approximate methods

The goal of the circuit design research developed in this thesis is to concurrently achieve both correct logic functionality and utilize a minimal number of logic gates. Since the proposed approach evaluates expressions based on user definitions, any type of logic i.e. the multiple-valued or Boolean logic, could be implemented (theoretically) with ease. Only limited Boolean logic applications by evolutionary learning methods have been demonstrated by other authors [Coon94, Koza92, Koza94, Koza99]. This will change with the creation of quantum computers, where an arbitrary GA-like problem or Constraint Satisfaction Problem would be directly solvable in hardware, using our proposed here approach (or possible improved future approaches that will be derived in future from our approach or competing approaches of Hugo De Garis or Han and Kim).

In addition to the exact quantum synthesis meta-algorithm, we created a heuristic search algorithm variant ECPS to be simulated in software. This simulator has only a limited potential. It is well-known that the simulation of quantum computers in standard computers is, from its very principle, very inefficient [Feynman]. With the limitations of the current computers, the experimental results of one preliminary version show that the logic expressions from this technique can produce better than 88% coverage of minterms in the given truth tables, but unfortunately the method cannot guarantee complete (100%) coverage. Thus, the method cannot be used for the

design of arbitrary quantum circuits, but can still be 100% convergent for some functions and is useful for Data Mining, Machine Learning, Knowledge Discovery, and robotics applications. The same method implemented in a future quantum computer may reach 100% convergence. There is another point here. Normally we assume 100% correct hardware but it is not sure at this time that the future self-repairing fault-tolerant computers will require 100% coverage for many problems. This means a quantum system composed of incorrect (faulty) components may operate correctly as the entire system. The grateful degradation allows the system to work correctly even if some percent of logic gates do not work correctly. Concluding on this aspect, the requirement for logic synthesis may change in future, making our algorithms useable with even less than 100% convergence.

16.10. Search with many strategies and heuristics

The approach to develop the QSPS/ECPS software/hardware uses a limited amount of humanly-designed, application specific heuristics. (The goal was for instance to develop the best GRM minimizer, producing minimizations better than those of a highly optimized human-designed algorithm by Debnath and Sasao [Debnath95, Debnath96].) In this algorithm design, the heuristics are used basically as generators of many good starting points for searches. Thus, logic heuristics are utilized to intelligently limit the size of the search space, while the search utilizes the quantum mechanics properties as a model from which the detailed logic minimization

algorithms are determined. This is an evolutionary mechanism but it is unlike the Darwinian, Lamarckian or Baldwinian variants of evolution. The heuristics applied here include the following: the addition of a "best-bounds" local heuristic search, logic theory applied in an expression specific manner which limits the minimization search space [Csanky93], starting with a population of expected good (rather than randomly generated) solutions (such as various "seeds" of ESOP circuits), and simultaneously investigating multiple solution trajectories per expression (all minimal cost expansion trees). We have simulated over 110 MCNC benchmarks [MCNC91], to check if these heuristics combined with evolutionary approaches show equal or reduced performance with that of the pure Genetic Algorithm. The ECPS (Extended Cybernetic Problem Solver) Algorithm was invented on base of comparisons as a better learning technique, as it incorporates a number of human and automated learning mechanisms and is more widely applicable. It was done even without utilizing the real power of quantum computing. I believe that ECPS can be further improved when more applications will be investigated.

16.11. The implemented "Extended Cybernetic Problem Solver" versus the general quantum search model from the thesis

The development of the Extended Cybernetic (Multi-Strategic Learning) Problem-Solving (ECPS) Algorithm design was based on the assumption that "any

combinatorial problem can be solved by searching some space of states". Many algorithms in this thesis confirm this assumption.

The solutions in ECPS are achieved with state-space-based serial-parallel search mechanisms. All software implemented by me in this thesis is only serial. Similarly only single-processor quantum computers were simulated by me, not parallel quantum computers. This task remains for the future work. The search mechanisms discussed are so general that they apply to both normal and quantum search domains. Herein, any type of search methodology may be employed and enhanced with corresponding learning mechanisms with which the search can find better solutions in less time, within the given state space size. The Human-designed Expert Systems often work well, but are limited in their practical applications. The traditional pure search strategies (breadth first, depth first, branch-and-bound, etc.) are comprehensive, guaranteeing an optimal solution from the solution space, but are (usually) prohibitively memory- and time- consuming. Quantum search combined with classical search in multi-level hierarchical game-like strategies gives the promise to be superior both to classical blind search, classical heuristic-dominated search and quantum "non-informed" search.

As the previous researches have shown, the heuristic search methods of Genetic Algorithms/Programs, while providing a less thorough search of solution space, may

find quasi-optimal solutions as local optima in the search space. These algorithms are however unable to find other, better solutions. Further, the GA/GP approaches have limitations of size, computation time, and solution optimality. They give also no explanation of the design methodology or transferable (scalable) rules of generalization. But they have two great assets: generality and ease of creation and use.

Another general approach is functional decomposition such as Ashenhurst Curtis decomposition. The Functional Decomposition research in our group [Files97, Files98, Files98a, Perkowski97d], while creating good solutions, is not easily tunable to all technologies and specifically I do not know how to adopt it to reversible circuits. It can be however observed that the Ashenhurst-Curtis decomposition can be used itself as the learning method in the space of search parameters [Slagle70, Samuel59, Samuel67] in ECPS. This has been not implemented but is a possible future research direction.

Based on previous literature one can thus state that both the complete and incomplete search strategies are generally unsatisfactory for producing a multi-purpose problem solving technique for practical applications. Therefore, human expertise must be combined with search mechanisms, for the development of efficient problem-solving methods, rather than as expert tools (search methods) to re-invent new problem-solving mechanisms "from scratch". The ECPS Algorithm was designed to use

synergism to incorporate any type of learning, including both pure and heuristic search strategies. The techniques are combined to form a superset, intelligent "toolbox" of various learning methodologies into a single algorithm. (The search methods currently implemented are a GA, random search, "simple" (bit-flipping) search, breadth-first, A*, best bound and depth-first strategies. However, any other methods can be easily added.)

This ECPS Algorithm has the capability of automatically producing customized, problem specific solutions which may combine a number of different solution search strategies for problem solution. Further, ECPS builds on the strengths, and thus reduces the weaknesses, of different well-known search methodologies. Elements of game theory were introduced into the ECPS Algorithm, as the different search methods first compete for efficiency and then cooperate to produce the most direct route to producing a quality solution. As the standard input and output formats are netlists (input may be a truth table, output may be a quantum array), the combinatorial logic may also be depicted graphically as a tree, decision diagram, K-map, or algebraic form, aiding the user in modifications of strategies, heuristic development, visualization of data, and the optimization process [Perkowski99c] (see chapter 7).

By combining search strategies through game theory [Samuel59, Samuel67], involving both competition and cooperation, as well as adding feedback to traditional methods,

the technique can be further improved. It should always produce results no worse than the standard techniques and often much better, as the strengths of all extended methods will be utilized. In contrast to standard "Darwinian" genetic/evolutionary methods, this approach learns (at least in theory) from problem to problem by generalizing successful problem-solving strategies.) We should however further study these aspects of ECPS design.

Even in the current ECPS the experienced human designer/problem-solver is not out of the loop. He can collaborate with the systematic and evolutionary components of the program, providing high-level feedback. The applications of the ECPS Algorithm indicate, on some problems, substantial performance improvements with this new algorithm versus other methods. For instance, the results of the GRM minimization may be compared to those of Debnath and Sasao [Debnath95, Debnath96]. Thus, Debnath and Sasao developed software which can minimize GRM functions with a larger number of variables and multi-outputs, but this software is only applicable to the minimization of completely specified functions. In contrast, the ECPS software is a general solution search method that can be employed for any logic problem, complete or not. Applying the ECPS to the GRM minimization problem, it is capable of solving both completely and incompletely specified functions. The minimization of incompletely specified functions is well known to be more difficult than the minimization of the completely specified forms [Dill01]. Our goal was that the

application of the ECPS to the incompletely specified GRM minimization problem will produce results that are superior to those previously achieved by Dill and Perkowski [Dill01] with the iGRMMIN software, the first and only other software designed for this application. The ECPS utilizes both a human designed heuristic and a genetic algorithm, and it employs many different non-quantum search techniques. QSPS adds quantum searches.

16.12. Arguments for AND/EXOR logic in binary quantum applications

One may ask why I used AND/EXOR logic as the fundament of all designs and software in this thesis? The answer was partially given in previous chapters and can be concluded as follows:

1. AND/EXOR logic is more natural for reversible and quantum circuits than other types of logic.
2. Some ideas of previous authors who worked on AND/EXOR logic can be used.
3. The AND/EXOR logic is relatively easily generalizable to Multiple-Valued logics – from GF(2) to GF(n).
4. The AND/EXOR logic is highly testable.

16.12.1. Galois Fields Logic for quantum circuits

It is interesting to note that the binary AND-EXOR logic represents a special case of the Galois Field Logic [Batisda84, Stewart89, Winter74, Edward93, Bell66] GF(k), where the radix k=2. Thus, Galois Field (Galois for short) logics can be viewed as a generalization of a subset of Boolean Logic because the Galois Field mathematical operations are applicable for multiple-valued logic values, over any finite field. Using of Galois logic allows us to have a mathematically beautiful synthesis theory and apply mathematics. It allows also to create a general theory for every finite field $GF(N^k)$. Thus, as the quantum logic is based on Pauli X, Y and Z rotations, the concept of rotation is the most natural for quantum logic at the low level. Rotation leads to modulo counting and modulo addition and is the operation used for add operator in every radix of MV logic being a prime number. Composition of rotations in various axes X, Y, Z leads to algebraic structures called rings. Realization of GF operations for GF(3) and GF(4) has been shown to be not too complex in the literature. All these are good arguments for AND/EXOR logic and its generalizations such as Modulo-based, ring-based and Galois Field based logics.

But it is still questionable if Galois Logic is the best logic from the low-level quantum circuit realization point of view. This problem is left open in my dissertation as it is a subject of the Ph.D. of Dipal Shah [Shah07]. Let us only observe that the Galois logic is only one of many possible extensions of AND/EXOR logic for d-level quantum

circuits, extensions that proved to be dominant in quantum circuits in year 2007. The other is the Controlled-Gate-Logic (our own name as this logic is not known from literature) which seems to be the best offer of current technologies. We did not discuss all generalizations of all methods proposed in this dissertation to the multiple-valued logics in chapter 10. We can refer the interested reader to the literature on the subject [Shah07, Giesecke07]. I agree, however, that these are only heuristic and analogy-based arguments for AND/EXOR logic to be used in quantum. Further research is still necessary. Hopefully it is being done in PSU quantum group of PSU Mathematics and ECE departments in dissertations of Faisal Khan, Ahmed Aden, Martin Lukac, Dipal Shah and Nouraddin Alhagi.

16.12.2. Highly Testable Quantum Circuits

It is well-known that the AND-EXOR logic, especially two-level logic, is the most highly testable of all digital logic structures [Perkowski97a] used in classical logic. In EXOR cascade [Reddy72] any stuck-at-1 or stuck-at-0 fault changes the polarity of signal seen at the output. It makes EXOR logic perfect for testability. Although the stuck-fault model is not good for quantum circuits [Biamonte04], the rotation model (like inserting an inverter to a quantum wire) also changes the output polarity. So, high testability methods can be rather easily adapted from classical to quantum logic [Biamonte05d, Perkowski07]. Therefore I believed that the AND-EXOR logic is a good candidate for permutative quantum design, where the fault-tolerance and high

testability issues are especially important [Kalay99, Kalay99a, Drechsler99, Perkowski99a, Reddy72, Sarabi93].

The recent top results in testing ESOP circuits (these circuits include the GRM, FPRM, and PPRM forms as special cases) were given by Kalay et. al [Kalay99a] as:

"...a simple, universal test set which detects all single stuck-at faults in the internal lines and the primary inputs/outputs of the realization... (the) circuit realization requires only two extra inputs for controllability and one extra output for observability. The cardinality of our test set for an n input circuit is (n + 6)..."

The test set developed by Kalay et al [Kalay99a] can also be very successfully applied to Built-in Self-Test (BIST) applications, and the concept of quantum BIST has been developed by Biamonte and Perkowski. It was then a hope, while developing the AND/EXOR reversible circuits in this dissertation that the methods from [Kalay99c] and Biamonte and Perkowski [Biamonte04, Biamonte05, Biamonte05a, Perkowski05] can be expanded to a wider category of quantum oracles. It is evident that the two-level AND-EXOR logic family is highly testable with a very limited number of test vectors assuming the classical "stuck-at" fault model. Again, because of the similarity of these circuits to quantum circuits (specially some structures) and because of the current understanding of fault models for quantum computing research of (Biamonte and Perkowski [Biamonte05c], Hayes and Ralf [Ralf05]), I believed when starting the work on this dissertation, that all these methods can be extended to multiple-valued

Galois and Controlled-Gate Quantum Logics. These other results can be found in [Shah07]. The high-testable EXOR-logic based circuits that originate from Reddy and were next extended by Sasao [Sasao95g] and Kalay et al [Kalay99c] were further extended in classical logic by Bhattacharya et al [Bhattacharya1] to bridging faults and to other AND/EXOR structures. The general idea is, the more EXORs and group-based operators, the more is the circuit testable.

Concluding, currently the basic research and CAD tool development for various AND-EXOR forms is increasingly at the forefront of the classical logical research as documented by many papers in RM, ISCAS, ULSI, ISMVL symposia and DAC conferences. The existing development in quantum circuit design area is dominated by this kind of logic, but so far there have been no any work besides book by Anas Al-Rabadi [Al-Rabadi04] based on his Ph.D. from PSU [Al-Rabadi02a] that would try to unify all the existing AND/EXOR approaches with respect to reversible logic synthesis, and especially for quantum realized oracles and not just the reversible logic oracles.

All together I believe that I solved most of the problems that I originally proposed to be done in this thesis. However, in the course of writing the thesis several new problems appeared and only some of them have been solved. The other problems will be further developed and solved in PSU Quantum Computing group and by me with my students in Bangladesh.

References

[Abe02] E. Abe , T. D. Ladd, J. R. Goldman, F. Yamaguchi, Y. Yamamoto, K. M. Itoh, "All-Silicon Quantum Computer", Phys. Rev. Lett. 89, 017901 (2002).

[Agrawal04] A. Agrawal and N. K. Jha. "Synthesis of reversible logic," in Proc. *DATE*, Paris, France, pp. 710- 722, February 2004.

[Aharonov03] Aharonov D., A. Ta-Shma A.: *Adiabatic Quantum State Generation and Statistical Zero Knowledge.* In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing. ACM Press, New York, 2003, pp. 20–29.

[Ahn00] J. Ahn, T.C. Weinacht, P. H. Bucksbaum, "Information Storage and Retrieval through Quantum Phase", Science 287, 463 (2000).

[Akers59] S.B. Akers, "On a Theory of Boolean Functions," J. of SIAM, vol. 7, pp. 487-498, 1959.

[Akers88] S.B. Akers, "On the use of linear assignment algorithm in module placement," 25 Years of Electronic Design Automation, 1988, pp. 218-223.

[Alexits61] G. Alexits, Convergence Problems of Orthogonal Series, (New York: Pergamon Press, 1961).

[Alhagi08] Alhagi, N., "Synthesis of Hybrid Reversible Cascades for Relational Input-Output Specifications", Phd Thesis, preparation, 2008, Portland State University, USA.

[Allen05] J. Allen, J. Biamonte and M. Perkowski, "ATPG for Reversible Circuits using Technology-Related Fault Models," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005.

[Almaini89] A. E. A. Almaini., Electronic Logic Systems, second edition (Englewood Cliffs, NJ: Prentice-Hall), Chap. 12, 1989.

[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

[AlRabadi02] A. Al-Rabadi, L. Casperson, M. Perkowski, and X. Song, "Canonical representation for Two-Valued Quantum Computing", Proc. Fifth Intern. Workshop on Boolean Problems, pp. 23-32, September 19-20 2002, Freiberg, Sachsen, Germany.

[Al-Rabadi01] 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

[Al-Rabadi02] A. Al-Rabadi, "Novel Methods for Reversible Logic Synthesis and Their Application to Quantum Computing", Ph. D. Thesis, PSU, Portland, Oregon, USA, October 24, 2002.

[AlRabadi04] A.N. Al-Rabadi, "Reversible Logic Synthesis", 2004, Springer, ISBN 3-540-00935-3

[AlRabadi05] A. N. Al-Rabadi and M. Perkowski, "New Families of Reversible Expansions and their Regular Lattice Circuits," *Journal of Multiple-Valued Logic and Soft Computing (MVLSC),* U.S.A., Volume 11, Number 3-4, 2005.

[Bae07] J. H. Bae, Ch. B. Bae, G. B. Lee, D. H. Kim, M.A. Perkowski, M.H.A. Khan "Minimization of Ternary and Mixed Binary-Ternary Permutative Quantum Circuits". *Report PSU,* 2007.

[Banks71] E. Banks. Information Processing and Transmission in Cellular Automata. MIT PhD. Thesis (1971)

[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.

[Batisda84] J.R. Batisda, Field Extensions and Galois Theory, (New York: Cambridge University Press, 1984).

[Beach03] Beach G., Ch. Lomont, Ch. Cohen, *Quantum Image Processing.* In: Proc. 32nd Applied Imagery Patter Recognition Workshop (AIPR'03), Washington DC, 2003, p. 39.

[Behrman96] E. Behrman, J. Niemel, J. Steck, S. Skinner, " A Quantum Dot Neural Network", Proceedings of the Workshop on Physics of Computation, pp. 22-24, 1996.

[BrassardGalois03] S. Beauregard, G. Brassard, J. M. Fernandez, Quantum Arithmetic on Galois Fields, *quant-ph/0301163.*

[Bell66] A. W. Bell, Algebraic Structures, (New York: John Wiley & Sons, Inc., 1966).

[Bennett73] C. H. Bennett, "Logical Reversibility of Computation", *IBM Journal of Research and Development*, 17, 1973, pp. 525-532.

[Benioff98] Benioff P., *Quantum Robots and Environments.* Phys. Rev. A 58, Issue 2, August 1998, pp. 893–904.

[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.

[Bennett93] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels" Phys Rev Lett, Issue 13 – March 1993, pp.1895-1899.

[Bennett96] Bennet CH, Brassard G, Popescu B, Smolin JA, Wotters WK (1996) Phys Rev Lett 76:722

[Beers98] G. E. Beers, K. L. John, "Novel Memory Bus Driver/Receiver Architecture for Higher Throughput", Proc. of the IEEE Int. Conf. On VLSI Design, pp. 259-264, 1998.

[Bentlet99] Bentlet and J. Bentley, Evolutionary Design by Computers, (San Francisco, California: Morgan Kaufmann Publishers, 1999).
[Berry97] J. A. Berry and G. Linoff, Data Mining Techniques: For Marketing, Sales, and Customer Support, (New York: John Wiley & Sons, Inc., 1997).
[Bhattacharya1] I Bhattacharyya, K Verma, GK Andagunda, A Sethi, "Reversible Computation and Quantum Computing", http://home.iitk.ac.in/~dgoswami/notes .
[Biamonte04] J. Biamonte, M. Perkowski, Principles of Quantum Fault Diagnostics, McNair research Journal, Issue 1, Volume 1, 2004
[Biamonte05] J. Biamonte, M. Perkowski, "Automated Test Pattern Generation for Quantum Circuits," McNair Research Journal, Vol. 1, Issue 1, 10 pages, 2005
[Biamonte05a] J. Biamonte, M. Perkowski, "Tricks to validate quantum switching networks", poster and presentation, Proc. of KIAS-KAIST 6th Workshop on Quantum Information Science, Seoul, Korea, pp. 9, August 22nd - 24th, (2005)
[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.
[Biamonte05c] J. Biamonte, J. Allen, D. Pierce, F. Khan and M. Perkowski, "Automated Test Set Generation for Quantum Circuits," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005.
[Biamonte07] J. Biamonte and M. Perkowski, "A Quantum Test Algorithm," *Submitted to IEEE Transactions on Computers and quant-ph/0501108.*
[Blais00] A. Blais, A. M. Zagoskin, "Operation of universal gates in a solid-state quantum computer based on clean Josephson junctions between d-wave superconductors", Phys. Rev. A 61, 042308 (2000).

[Boole54] G. Boole, An Investigation of the Laws of Thought, (London: Walton, 1854). (Reprinted by Dover Books, New York, 1954.)
[Boyer96] M. Boyer, G. Brassard, P. Hoyer, and A. Tapp, "Tight bounds on quantum searching," *Proceedings of PhysComp,* 1996.
[Breazeal02] Breazeal C., *Designing Sociable Robots.* MIT Press, 2002.
[Bronco95] A. Bronco et al., "Elementary Gates For Quantum Computation", *Physical Review A* 52, 1995, pp. 3457-3
[Britton06] J. Britton, D. Leibfried, J. Beall, R. B. Blakestad, J. J. Bollinger, J. Chiaverini, R. J. Epstein, J. D. Jost, D. Kielpinski, C. Langer, R. Ozeri, R. Reichle, S. Seidelin, N. Shiga, J. H. Wesenberg, D. J. Wineland , "A microfabricated surface-electrode ion trap in silicon", 2006, arXiv:quant-ph/0605170v1.
[Brown90] F.M. Brown, Boolean Reasoning: The Logic of Boolean Equations, (Boston, Massachusetts: Kluwer, 1990).
[Bruce02] J.W. Bruce, M.A. Thornton, L. Shivakumaraiah, P.S. Kokate, and X. Li, "Efficient Adder Circuits Based on a Conservative Reversible Logic Gate", *Proc. of the IEEE Computer Society Annual Symposium on VLSI,* Pittsburgh, Pennsylvania, April 2002, pp. 83-88.

[Brylinski01] J. L. Brylinski, and R. Brylinski, "Universal Quantum Gates," *arXiv: Quant-ph/0108062*
[Buller03a] A. Buller, M. Perkowski, "Cellular Automata realization of Regular Logic", Booklet of 12th International Workshop on Post-Binary ULSI Systems, *May 16, 2003, Meiji University, Japan, pp. 53 -- 60.*
[Buller03] A. Buller, M. Perkowski, "Evolved Reversible Cascades Realized on the CAM-Brain Machine", IEICE Proceedings of NASA\DoD conference on Evolvable Hardware,2003, pp. 246-251.
[Bullock05] S.S. Bullock, D.P. O'Leary, and G.K. Brennen, "Asymptotically Optimal Quantum Circuits for d-level Systems, " *Phys. Rev. Lett.* 94, 230502, 2005.
[Burns98] M. Burns, M. Perkowski, L. Jozwiak, and S. Grygiel, "An Efficient and Effective Approach to Column-Based Input/Output Encoding in Functional Decomposition", Proc. of the 3rd International Workshop on Boolean Problems, Freiberg University of Mining and Technology, Institute of Computer Science, September 17-18, 1998, pp. 19-29.

[Brassard04] G. Brassard, "Quantum Communication Complexity: A Survey", 34th International Symposium on Multiple-Valued Logic (ISMVL'04), 2004, Canada, pp.56.

[Brassard97] G. Brassard, P. Høyer, " An exact quantum polynomial-time algorithm for Simon's problem", Proceeding of the Fifth Israeli Symposium on Theory of Computing and Systems, IEEE Computer Society Press, June 1997, pp. 12—23.
[Brassard98] G. Brassard, "New horizons in quantum information processing", Proceedings of this ICALP Conference, 1998.
[Brassard98a] G. Brassard, P. Høyer, A. Tapp, "Quantum counting" Lecture Notes in Computer Science 1443 (1998), 820+.

[Brayton87] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. R. Wang, "MIS: A Multiple-Level Logic Optimization System", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, CAD-6, pp. 1062-1081, Nov. 1987.

[Chang98] C. H. Chang and B. J. Falkowski, "Adaptive Exact Optimisation of Minimally Testable FPRM Expansions", IEE Proc. – Computers and Digital Techniques, Nov. 1998, Vol. 145, Issue 6, p. 385

[Chang99] C.H. Chang, and B.J. Falkowski, " NPN Classification using weight and literal vectors of Reed-Muller expansion" IEEE Electronics Letters, 13th May 1999, Vol. 35 No. 10

[Chen02] G. Chen, S.A. Fulling, and J. Chen, Generalization of Grover's Algorithm to Multi-object Search in Quantum Computing, Part I: Continuous Time and Discrete Time, quant-ph/0007123. Also in Chapt. 6 of " Mathematics of Quantum Computation", edited by R. K. Brylinski and G. Chen, CRC Press, Boca Raton, Florida, 2002, pp. 135-160.

[Cheng05] Cheng Fu, and B.J. Falkowski, "Ternary Fixed Polarity Linear Kronecker Transforms and their Comparison with Ternary Reed-Muller Transform," *Journal of Circuits, Systems, and Computers*, Vol. 14, No. 4 (2005) pp. 721–733.

[Chuang95] I. Chuang, R. Laflamme, P. Shor, W. Zurek "Quantum Computers, Factoring, and Decoherence", Arxiv preprint quant-ph/9503007, 1995 - arxiv.org.

[Chuang98] I. Chuang, N. Gershenfeld, M. Kubinec, "Experimental Implementation of Fast Quantum Searching" ,Physical Review Letters, Issue 15 – April 1998 , pp. 3408 – 3411.

[Cleve98] R. Cleve, W. van Dam, M. Nielsen, A. Tapp, "Quantum Entanglement and the Communication Complexity of the Inner Product Function",International Conference, QCQC'98, Palm Springs, California, USA, February 1998.

[Cohn62] M. Cohn, "Inconsistent Canonical Forms of Switching functions", IRE Trans. On Electr. Comp., Vol. EC-11, pp. 284-285, 1962.

[Coon94] B. W. Coon, "Circuit Synthesis through Genetic Programming", Genetic Algorithms at Stanford 1994, Compiled by John R. Koza, (Stanford, California: Stanford University, 1994).

[Cordone01] R. Cordone, F. Ferrandi, D. Sciuto, R. W. Calvo, "An efficient heuristic approach to solve the unate covering problem", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 20, Issue 12, Dec 2001, pp. 1377 - 1388

[Cory97] D.G. Cory, A.F. Fahmy, and T.F. Havel, Nuclear magnetic resonance spectroscopy: an experimentally accessible paradigm for quantum computing, in Proc. of the 4th Workshop on Physics and Computation (Complex Systems Institute, Boston, New England) 1996 Science 275, 350 (1997).

[Csanky93] L. Csanky, M. Perkowski, I. Schaefer, "Canonical Restricted Mixed-Polarity Exclusive-Or Sums of Products and the Efficient Algorithm for their Minimization," IEE Proceedings, Pt.E, Vol. 140, No. 1, pp. 69 - 77, January 1993.

[Curtis04] Curtis, E., Perkowski, M. "A transformation based algorithm for ternary reversible logic synthesis using universally controlled ternary gates", *Proc. IWLS 2004*, Tamecula, California, USA, 2-4 June 2004. pp. 345 – 352.

[Curtis07] E. Curtis, and M. Perkowski, Minimization of Ternary Reversible Logic Cascades using a Universal Subset of Generalized Ternary Gates, accepted to *International Journal on Multiple-Valued Logic and Soft Computing,* Svetlana Yanushkevich, editor. ISSN 1542-3980. ISI.

[Das03] R. Das, A. Mitra, V. Kumar and A. Kumar, "Quantum information processing by NMR: Preparation of pseudo pure states and implementation of unitary operations in a single-qutrit system", *arXiv-quant-ph/0307240v1*, 31 July 2003.

[Davio78] M. Davio, J. P. Deshamps, A. Thayse, "Discrete and Switching Functions", McGraw Hill International, 1978

[Debnath95] D. Debnath and T. Sasao, "GRMIN: A Heuristic Simplification Algorithm for Generalized Reed-Muller Expressions", IFIP WG 10.5, Proc. of the Workshop on Applications of the Reed-Muller Expansion in Circuit Design, 27-29 August 1995, Makuhari, Chiba, Japan.

[Debnath96] D. Debnath and T. Sasao, "GRMIN2: A Heuristic Simplification Algorithm for Generalized Reed-Muller Expressions, IEE Proc. Comput. Digit. Tech., 143 (6) (1996).

[Debnath98] D. Debnath, "On the Minimization of AND-EXOR and AND-OR-EXOR Networks", Diss. Kyushu Institute of Technology, Japan, March 1998.

[DeGaris92] H. de Garis, "Artificial Embryology: The Genetic Programming of an Artificial Embryo", Dynamic Genetic, and Chaotic Programming, Branko Soucek and the IRIS Group, (New York: John Wiley & Sons, Inc. 1992).

[Denler04] N. Denler, B. Yen, M. Perkowski and P. Kerntopf, "Synthesis of Reversible Circuits from a Subset of Muthukrishnan-Stroud Quantum Realizable Multi-Valued Gates", *Proceedings of IWLS 2004*, Tamecula, California, USA, 2-4 June 2004.

[Denler04a] N. Denler, B. Yen, M. Perkowski, and P. Kerntopf, "Minimization of Arbitrary Functions in a New Type of Reversible Cascade built from Quantum-Realizable Generalized Multi-Valued Gates" , *Proc. IWLS 2004.* pp. 321 – 328.

[Deutsch89] Deutsch, D. Quantum Computational networks. Proc. R. Soc. Lond. A 425 (1989) 73-90.

[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, K. Ganguly, R. J. Safranek, and M. A. Perkowski, "A New Linearly Independent, Zhegalkin Galois Field Reed-Muller Logic", Portland State University Department of Electrical and Computer Engineering Report, 1997.

[Dill97b] K.M. Dill, J. Herzog, and M. Perkowski, "Genetic Programming and its Application to the Synthesis of Digital Logic", Proc. of the PACRIM '97 Conference, Victoria, Canada, Aug. 20-22, 1997, (Piscataway, New Jersey: IEEE 1997).

[Dill97c] 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.

[Dill98] K. M. Dill and M. A. Perkowski, "Evolutionary Minimization of Generalized Reed-Muller Forms", Proc. of the International Conference on Computational Intelligence and Multimedia 1998 (ICCIMA'98), Monash University, Churchill, Vic., Australia, 9-11, February 1998.

[Dill01] K.M. Dill, and M. Perkowski, "Baldwinian Learning utilizing Genetic and Heuristic for Logic Synthesis and Minimization of Incompletely specified data with Generalized Reed-Muller (AND-EXOR) forms", Journal of System Architecture 47, Issue 6, 2001, pp. 477-489.

[Dong05] D. Dong , Ch. Chen, Ch. Zhang, Z. Chen, "*An Autonomous Mobile Robot Based on Quantum Algorithm*", Springer Berlin / Heidelberg , Volume 3801/2005.

[Dong06] D. Dong , Ch. Chen, Ch. Zhang, Z. Chen, "Quantum robot: structure, algorithms and applications", Robotica, Cambridge University Press(2006), 24, pp. 513-521

[Disman96] M. Disman, "Stalking the Chameleon Computer", Computer & Communications OEM Magazine, Vol. 3, No. 23, (December/January 1996), pp. 67-73.

[Drechsler96] Rolf Drechsler, Bernd Becker, Nicole Göckel, "A Genetic Algorithm For Minimization Of Fixed Polarity Reed-Muller Expressions," In *IEE Proceedings Computers and Digital Techniques*, Vol. 143, pp. 364-368, 1996

[DiVincenzo00] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation", Experimental Proposals for Quantum Computation, (2000), arXiv:quant-ph/0002077

[Drechsler97] R. Drechsler, "Evolutionary Algorithms for Computer-Aided Design of Integrated Circuits Tutorial", Genetic Programming 1997 Conference, Stanford University, July 13, 1997

[Drechsler99] R. Drechsler, H. Hengster, H. Schaefer, J. Hartmann, and B. Becker, "Testability of 2-Level AND/EXOR Circuits", Journal of Electronic Testing, Theory and Application, (JETTA), 1999

[Dubrova96] E. V. Dubrova and J. C. Muzio, "Testability of Generalized Multiple-Valued Reed-Muller Circuits", Proc. of the 26th International Symposium on Multi-Valued Logic, IEEE, 1996, pp. 56-61.

[Dubrova97] E. V. Dubrova, "Boolean and Multiple-Valued Functions in Combinational Logic Synthesis", Ph.D._Thesis, University of Victoria, Canada, 1997.

[Dubrova01] E. Dubrova, Y. Jiang, R. Brayton, "Minimization of Multiple-Valued Functions in Post Algebra", *Proc. IWLS01*, pp. 132-138, June 2001.

[Dueck03] G. W. Dueck and D. Maslov, "Reversible function synthesis with minimum garbage outputs," in Proc. 6th International Symposium on Representations and Methodology of Future Computing Technologies, Trier, Germany, pp. 154-161, March 2003.

[Dueck03a] G.W. Dueck and D. Maslov, "Garbage in Reversible Designs of Multiple-Output Functions," Proc. RM 2003, pp. 162 – 170.

[Dueck86] G. W. Dueck and D. M. Miller, "A 4-Valued PLA Using the MODSUM", Proc. of the 16th International Symposium on Multi-Valued Logic, May 1986, pp. 232-240.

[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.

[Durf03] Durt, N. J. Cerf, N. Gisin and M. Zukowski, "Security of Quantum Key Distribution with Entangled Qutrits," *Phys. Rev. A* 67, 012311, 2003, also *quant-ph/0207057*.

[Dwave07] http://dwave.wordpress.com/2007/01/19/quantum-computing-demo-announcement/. Look also to many materials linked from this webpage.

[Edward93] H. M. Edward, Galois Theory, (New York: Springer-Verlag, 1993).

[Einstein35] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Phys. Rev.*, vol. 47, no. 10, pp. 777–780, May 1935.

[Eisert99] J. Eisert, M. Wilkens, M. Lewenstein, "Quantum Games and Quantum Strategies" Physical Review Letters 83, 3077 - 3080 1999.

[Falkowski97] B.J. Falkowski and C.H. Chang, "Properties and Methods of Calculating Generalized Arithmetic and Adding Transforms," *IEE Proc. Circuits, Devices, and Systems,* vol. 144, no. 5, pp. 249-258, Oct. 1997.

[Falkowski97a] B.J. Falkowski, V.P. Shmerko, and S.N. Yanushkevich, "Arithmetical Logic—Its Status and Achievements," *Proc. Int'l Conf. Applications of Computer Systems,* pp. 208-223, Szczecin, Poland, Nov. 1997.

[Falkowski03] B.J. Falkowski, C.C. Lozano, "Generation and properties of fastest transform matrices over GF (2)", Circuits and Systems, 2003. ISCAS'03, Volume: 4, pp.IV-740- IV-743 vol.4, ISBN: 0-7803-7761-3

[Falkowski03a] B.J. Falkowski, C.C. Lozano, "Polynomial expansions over GF(3) based on fastest transformation", Proceedings. 33rd International Symposium on Multiple-Valued Logic, 2003, pp. 40- 45, ISBN: 0-7695-1918-0

[Falkowski03b] B. J. Falkowski, F. Cheng, "Fast linearly independent ternary arithmetic transforms", Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS 03, Volume 4, Issue , 25-28 May 2003, pp. IV-560 - IV-563 vol.4

[Falkowski05] B.J. Falkowski, C. Fu, "Fastest classes of linearly independent transforms over GF(3) and their properties", Computers and Digital Techniques, IEE Proceedings, 2005, Volume: 152, Issue: 5, pp. 567- 576, ISSN: 1350-2387.

[Fan07] Fan Y., "Generalization of Deutsch-Jozsa algorithm to Multiple-Valued Quantum Logic", Proc. ISMVL 2007, http://ismvl07.ifi.uio.no/.

[Farhi98] E. Farhi, S.Gutmann, "Quantum computation and decision trees", Phys. Rev. A 58, 915 - 928 (1998).

[Fei02] X. Fei, D. Jiang-Feng, S. Ming-Jun, Z. Xian-Yi, H. Rong-Dian, and W. Ji-Hui, "Realization of the Fredkin gate by three transition pulses in a nuclear

magnetic resonance quantum information processor," *Chinese Phys. Lett.*, vol. 19, no. 8, pp. 1048–1050, 2002.
[Feynman65] R. Feynman, R. Leighton, M. Sands, " The Feynman Lectures on Physics", v3, Addison-Wesley, 1965.
[Feynman82] R. Feynman, "Simulating physics with computers", Int. J. Theor. Phys., 21:467, 1982
[Feynman96] R. Feynman, "Feynman Lectures on Computation", Addison Wesley, 1996
[Files97] C. Files, R. Drechsler, and M. Perkowski, "Functional Decomposition of MVL Functions Using Multi-Valued Decision Diagrams", 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).
[Files98] C. Files and M. Perkowski, "An Error Reducing Approach to Machine Learning Using Multi-Valued Functional Decomposition", Proc. of the International Symposium on Multi-Valued Logic 1998, Fukuoka, Japan, May 26, 1998, (Piscataway, New Jersey: IEEE, 1998), pp. 167-172.

[Files98a] C. Files and M. Perkowski, "Multi-Valued Functional Decomposition as a Machine Learning Method", Proc. of the International Symposium on Multi-Valued Logic 1998, Fukuoka, Japan, May 26, 1998, (Piscataway, New Jersey: IEEE, 1998), pp. 173-178.

[Files02] A. P. Flitney, and D. Abbott, "Quantum version of Monty Hall problem," *Phys. Rev. A.* Vol. 65, 062318, 2002.

[Fredkin82] E. Fredkin and T. Toffoli, "Conservative logic", *Intern. J. Th. Physics,* 21, pp. 219-253, 1982.

[Fredkin03] E. Fredkin: "An introduction to Digital Philosophy", International Journal of Theoretical Physics, Volume 42, Number 2, pp 189-247 (2003).

[Fujiwara86] H. Fujiwara, "Logic Testing and Design for Testability", Computer Science Series, (Cambridge, Massachusetts: The MIT Press, 1986).
[Gamberger97] D. Gamberger and Nada Lavrac, "Conditions for Occam's Razor Applicability and Noise Elimination", Proc. of the 9th European Conference on Machine Learning, Prague, Czech Republic, April 23-25, 1997, (Berlin, Germany: Springer-Verlag, 1997).

[Gamst96] A. Gamst, " Some lower bounds for a class of frequency assignment problems", IEEE Transactions of Vehicular Technology, 35(1):8- 14, 1996.
[Garey77] M. Garey and D. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32:826-834, 1977.
[Garey79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.
[DeGaris92] H. de Garis, "Artificial Embryology: The Genetic Programming of an Artificial Embryo", Dynamic Genetic, and Chaotic Programming, Branko Soucek and the IRIS Group, (New York: John Wiley & Sons, Inc. 1992).

[DeGaris93] H. de Garis, "Evolvable Hardware: Genetic Programming of a Darwin Machine", Artificial Neural Nets and Genetic Algorithms: Proc. of the International Conference in Innsbruck, Austria, 1993, (New York: Springer-Verlag, 1993).

[Gershenfeld97] N.A. Gershenfeld and I.L. Chuang, Bulk Spin-Resonance Quantum Computation, Science 275, 350 (1997).

[Giesecke06] Giesecke N.: *Ternary Quantum Logic*. M.S. thesis, PSU, Dept ECE, 2006.

[Giesecke07] N. Giesecke, D.H. Kim, S. Hossain, M. Perkowski, (2007). Search for universal ternary quantum gate sets with exact minimum costs. *37th IEEE Int. Symp. On Multiple-Valued Logic (ISMVL 2007)*, Oslo, Norway, 14-15 May 2007. http://ismvl07.ifi.uio.no/

[Giesecke08] Giesecke, N., Hossain, S. Kim, D.H., Perkowski, M., " Search for Universal Ternary Quantum Gate Sets with Exact Minimum Costs," Embedded Software Design (Journal of System Architecture), 2008, The EUROMICRO Journal—Accepted with conditional revision.

[Gisin02] Gisin N, Ribordy G, Tittel W, Zbinden H (2002) Rev Md Phys 74:145

[Goldberg89] [Goldberg1 K1] [s1] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.

[Green91] D. H. Green, "Families of Reed-Muller Canonical Forms", International Journal of Electronics, 70 (1991), pp. 259-280.

[Greentree04] A. D. Greentree, S. G. Schirmer, F. Green, L. C. L. Hollenberg, A. R. Hamilton and R. C. Clark, "Maximizing the Hilbert Space for a finite Number of Distinquishable States," *Phys. Rev Lett.* 92, 097901, 2004.

[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.

[Grover98] L. Grover, "A framework for fast quantum mechanical algorithms", Proceedings of the thirtieth annual ACM symposium on Theory of computing, , Dallas, Texas, United States, pp. 53 - 62 ,1998 ,ISBN:0-89791-962-9

[Gruska99] J. Gruska, *Quantum computing*. Osborne/McGraw-Hill,U.S., 1999.

[Hagan96] M. T. Hagan, H. B. Demuth, and M. Beale, Neural Network Design, (New York: PWS Publishing Company, 1996).

[Hameroff96] S.R. Hameroff, R. Penrose, "Conscious events as orchestrated space-time selections", Imprint Academic, Journal of Consciousness Studies, Volume 3, Number 1, 1996 , pp. 36-53(18).

[Hameroff98] S.R. Hameroff, "Quantum computation in brain microtubules? The Penrose-Hameroff 'Orch OR' model of consciousness Philosophical Transactions", Volume 356, Number 1743/August 15, 1998, pp. 1869-1896.

[Hanson93] J. E. Hanson, Computational Mechanics of Cellular Automata. PhD Thesis, Physics Department, University of California, Berkeley, CA, 1993.

[Harata87] Y. Harata, Y. Nakamura, H. Nagese, M. Takigawa, and N. Takagi,"A High-Speed Multiplier Using a Redundant Binary Adder Tree,"IEEE J. Solid-State Circuits, vol. 22, pp. 28-34, Feb. 1987.
[Harison65] M.A.Harison, Introduction to Switching and Automata Theory, McGraw-Hill, 1965.
[Harodecki01] Horodecki P, Horodecki R (2001) Quant Inf Comp 1(1):45

[Hayward02] M. Hayward, Quantum Computing and Grover's Algorithm, 2002, http://alumni.imsa.edu/~matth/quant/473/473proj/node1.html
[Hemmi94] H. Hemmi, J. Mizoguchi, and K. Shimohara, "Development and Evolution of Hardware Behaviors", Artificial Life IV: Proc. of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Editors: Rodney A. Brooks and Pattie Maes, (Cambridge, Massachussetts: The MIT Press, 1994).

[Highes00] R.J. Highes, C. P. Williams, "Quantum Computing: The Final Frontier?" IEEE Intelligent Systems, Volume 15, Issue 5 (September 2000), pp. 10-18, ISSN 1541-1672

[Higuchi93] T. Higuchi, Niwa, Tanaka, Iba, de Garis, and Furuya, "Evolving Hardware with Genetic Learning: A First Step Towards building a Darwin Machine", From Animals to Animats 2: Proc. of the Second International Conference on Simulation of Adaptive Behavior, Editors: Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, (Cambridge, Massachusetts: The MIT Press, 1993).

[Higuchi94] T. Higuchi, H. Iba, and B. Manderick, "Evolvable Hardware", Massively Parallel Artificial Intelligence, Chapter 12, Editors: Hiroaki Kitano and James A. Hendler, (Menlo Park, California: AAAI Press / The MIT Press, 1994).

[Higuchi97] T. Higuchi, Evolvable Hardware Tutorial, Genetic Programming 1997 Conference, Stanford University, July 13, 1997.

[Higuchi97a] T. Higuchi and M. Iwata, Editors, Evolvable Systems: From Biology to Hardware, (Berlin, Germany: Springer-Verlag, 1997).

[Hirvensalo01] M. Hirvensalo, " An introduction to quantum computing", Current trends in theoretical computer science: entering the 21st centuary, 2001,World Scientific Publishing Co.,Inc. River Edge, NJ, USA, pp 643-663, ISBN 981-02-4473-8

[Hochbaum82] D. S. Hochbaum, "Approximation algorithms for the weighted set covering and node covering problems," SIAM J. Comput., Vol. 11, 1982, pp. 535-556.

[Holland92] J. H. Holland, "Genetic Algorithms", Scientific American, July 1992, pp. 66-72.

[Hopfield82] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proceedings of the National Academy of Scientists, v79, pp. 2554-2558, 1982.

[Hossain04] Hossain, S., Monirul Islam, Rezaul Bashar and Alamgir "Logical Reversibility Based on Reversible Computing Technology" Asian Journal of Information Technology, Volume 3 Number 4, 2004, pp. 241-244, ISSN: 1638-8831

[Hossain08] Hossain, S., Perkowski, M., "The affine gates and affine polarities for quantum arrays with small costs", 17th International Workshop on Post-Binary ULSI Systems, May 24, 2008, Dallas, Texas, USA.

[Hossain09] Hossain, S., "Classical and Quantum Search Algorithms for Quantum Circuits and Optimization of Quantum Oracles", Ph.D. Thesis, 2009, Portland State University, USA.

[Huang01] Huang Q., Yokoi K., Kajita S., Kaneko K., Arai H., Koyachi N., Tanie K.: *Planning Walking Patterns for a Biped Robot.* IEEE Trans. Rob and Autom, Vol. 17, No. 3, June 2001. pp. 280-289.

[Hung04] W. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Provably optimal reversible quantum logic synthesis via symbolic reachability analysis," in *Proceedings of DAC*, 2004.

[Hung06] W. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and systems*, vol. 25, no. 9, pp.1652–1663, 2006.

[Hurst85] S. L. Hurst, D. M. Miller and J. Muzio, "Spectral Techniques in Digital Logic", Academic Press, London, 1985.

[Hollenberg04] L.C.L. Hollenberg, A. S Dzurak, C Wellard, A. R Hamilton, "Charge-based quantum computing using single donors in semiconductors", Physical Review B 69, 2004 .

[Ibaraki76] T. Ibaraki, "Theoretial comparisons of search strategies in branch-and bound algorithms," Intern. Journal. Comp. Sci., 5, 1976, pp. 315-344.

[Ilachinski01] A. Ilachinski. Cellular Automata: A Discrete Universe. World Scientific publishing, Singapore, 2001.

[Iwama02] K. Iwama, Y. Kambayashi, S. Yamashita, "Transformation rules for designing CNOT-based quantum circuits," in Proc. *DAC*, New Orleans, LA, pp. 419-424, June 10-14, 2002.

[Jones98] J.A. Jones and M. Mosca, Implementation of a Quantum Algorithm on a Nuclear Magnetic Resonance Quantum Computer, J. Chem. Phys., 109, (1998) 1648

[Jones98a] J. Jones, R. Hansen and M. Mosca, "Quantum Logic Gates and Nuclear Magnetic Resonance Pulse Sequences," J.Magn.Resonance 135, pages 353-360, (1998), quant-ph/9805070.

[Jou93] Jer-Min Jou, Shiann-Rong, K. R. Chen , "Clique partitioning based integrated architecture synthesis for VLSI chips", Proceedings of International Symposium on VLSI Technology, Systems, and Applications, 1993, pp. 58-62, ISBN: 0-7803-0978-2

[Julstrom99] B. A. Julstrom, "Comparing Darwinian, Baldwinian, and Lamarckian Search in a Genetic Algorithm for the 4-Cycle Problem", Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference, GECCO'99, (S. Brave and A. S. Wu, Eds.), July 14-17, 1999, Orlando, Florida, pp. 134-138.

[Kalay98] Kalay, U., Hall, D., Perkowski, M. (1998). A minimal and universal test set for multiple-valued Galois field sum-of-products circuits. *Proc. 7th Workshop on Post-Binary ULSI Systems*, Fukuoka, Japan, May 1998, pp. 50-51.

[Kalay99] U. Kalay, N. Venkataramaiah, A. Mishchenko, D. Hall, and M. Perkowski, "Highly Testable Finite State Machines Based on Exor Logic", Proc. of the 7th IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, Victoria, B.C., Canada, August 23-25, 1999.

[Kalay99a] U. Kalay, M. Perkowski, and D. Hall, "A Minimal Universal Test Set for Self Test of EXOR-Sum-of-Products Circuits", IEEE Transactions on Computers, July 1999.

[Kalay99b] U. Kalay, M. A. Perkowski, and D. V. Hall, "Highly Testable Boolean Ring Logic Circuits", Proc. of the International Symposium of Multi-Valued Logic 1999, (ISMVL'99).

[Kalay99c]U. Kalay, D. V. Hall, and M. A. Perkowski, "Easily Testable Multiple-Valued Galois Field Sum-of-Products Circuits", Journal of Multiple-Valued Logic, January 1999.

[Kalay99c] U. Kalay, D. V. Hall, and M. Perkowski "Easily Testable Multiple-Valued Galois Field Sum-of-Products Circuits". *Journal on Multiple Valued Logic,* 2000, Vol. 5, pp. 507-528.

[Kari94] J. Kari, Reversibility and surjectivity problems of cellular automata. Journal of Computer and System Sciences, 48(1): pp. 149—182, 1994.

[Kari96] J. Kari. Representation of reversible cellular automata with block permutations. Mathematical System Theory, 29: pp. 47—61, 1996.

[Karp72] Karp, R.M.: "Reducibility Among Combinatorial Problems". Complexity of Computer Computation, Plenum Press, ed. Miller, pp. 85-103, New York, 1972.(One of most important early papers in complexity theory).

[Kempe02] J. Kempe, K.B. Whaley, "Exact gate-sequences for universal quantum computation using the XY-interaction alone," *Phys. Rev. A.* Vol. 65(5), 05230, 2002.

[Kerntopf04] P. Kerntopf, M. Perkowski and M.H.A. Khan, "On Universality of General Reversible Multiple-Valued Logic Gates"_Proceedings of ISMVL 2004, pp. 68-73.

[Kerntopf04b] P. Kerntopf. "A new heuristic algorithm for reversible logic synthesis," in Proc. *DAC*, pp. 834-837, June 2004.
[Kerntopf06] P. Kerntopf, M. Perkowski, M.H.A. Khan, Universality of ternary reversible gates. *Accepted to special issue of International Journal on Multiple-Valued Logic and Soft Computing,* Svetlana Yanushkevich, editor . ISSN 1542-3980.
[Khan05] F. Khan, and M. Perkowski, "Decomposition of Ternary Quantum Gates," *Proceedings of RM* 2005.

[Khan06] F. Khan, M. Perkowski, "Synthesis of multi-qudit hybrid and *d*-valued quantum logic circuits by decomposition" , Theoretical Computer Science ,Volume 367, Issue 3, 1 December 2006, pp. 336-346.
[Khan03] M.H.A. Khan, M. Perkowski, and P. Kerntopf, "Multi-Output Galois Field Sum of Products Synthesis with New Quantum Cascades," *Proceedings of 3rd International Symposium on Multiple-Valued Logic, ISMVL 2003*, 16-19 May 2003, Meiji University, Tokyo, Japan, pp. 146-153.

[Khan04] M. A. Khan, M. Perkowski, Ternary Galois field expansions for reversible logic and Kronecker decision diagrams for ternary GFSOP minimization. *Proc. of 34th IEEE Int. Symp. on Multiple-Valued Logic (ISMVL 2004)*, Toronto, Canada, 19-22 May 2004, pp. 58-67.
[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.

[Khan05a] M.H.A. Khan and M. Perkowski, "Quantum Realization of Ternary Parallel Adder/Subtractor with Look-Ahead Carry," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005. pp. 15-22.

[Khan05b] M.H.A. Khan, and M.Perkowski, "Quantum Realization of Ternary Encoder and Decoder," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005. pp. 23 – 27.

[Khan05c] M.H.A. Khan, M. Perkowski, M.R. Khan, and P. Kerntopf, Ternary GFSOP minimization using Kronecker decision diagrams and their synthesis with quantum cascades. *Journal of Multiple-Valued Logic and Soft Computing,* ISSN 1542-3980. *11*, 2005, pp. 567-602.
[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.
[Khan07]M. Khan, M. Perkowski, D.H. Kim and K. Dill, Investigating Learning Search Strategies for Exploring the Space of Local Equivalence Transformations for Optimization of Quantum Circuits, in preparation.

[Khlopotine02] A. Khlopotine, M. Perkowski, and P. Kerntopf, "Reversible logic synthesis by gate composition," Proceedings of IWLS 2002, pp. 261 – 266.

[Kim00] J. Kim, J-S. Lee and S. Lee, "Implementing unitary operators in quantum computation, *Physical Review A*, 032312, 2000.

[Kim06] D. H. Kim, Ch. Brawn, M. Sajkowski, T. Stenzel, T. Sasao, J.Allen, M. Lukac, and M. Perkowski, "Artificial Immune – Fuzzy System to control walking robot Hexor," *submitted to ISMVL 2006.*

[Kim06a] D. H. Kim, J. I. Park, M. Perkowski, "Intelligent Tuning of a PID Controller Using a Hybrid GA-PSO Approach", *submitted to Conference on Intelligent Control, 2006.*

[Klimov03] A. B. Klimov, R. Guzman, J. C. Retamal and C. Saavedra, "Qutrit quantum computer with trapped ions," *Phys. Rev. A* 67, 062313, 2003.

[Knill04] E. Knill, "Fault-tolerant postselected quantum computation: schemes," *quant-ph/0402171*, 2004.

[Knill05] E. Knill, "Quantum computing with realistically noisy devices", *Nature* 434, pp. 39-44 , 3 March 2005.

[Kohavi70] Z. Kohavi, "Switching and Finite Automata Theory," Mc Gruw-Hill, 1970.

[Kosko94] B. Kosko, "Fuzzy Systems as Universal Approximators", IEEE Transactions on Computers, Vol. 34, No. 11, 1994, pp. 1329-1333.

[Koza92] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, (Cambridge, Massachusetts: The MIT Press, 1992).

[Koza94] J. R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs, (Cambridge, Massachusetts: The MIT Press, 1994).

[Koza99] J. R. Koza, F. H. Bennett, and D. Andre, Genetic Programming III: Darwinian Invention and Problem Solving, (San Francisco, California: Morgan Kaufmann Publishers, 1999).

[Kruskal56] J. B. Kruskal, Jr. "On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem", Proceedings of the American Mathematical Society, Vol. 7, No. 1 (Feb., 1956), pp. 48-50.

[Kumer07] Kumar M., Year B., Metzger N., Wang, Y., Perkowski M., *Realization of Incompletely Specified Functions in Minimized Reversible Circuits.* Proc. RM 2007.

[Kumar07] M. Kumar, B. Year, N. Metzger, Y. Wang, and M. Perkowski, *Realization of Incompletely Specified Functions in Minimized Reversible Circuits.* Proc. RM 2007. http://ismvl07.ifi.uio.no/

[Kwiat99] P.G. Kwiat, J. R. Mitchell, P.D.D. Schwindt, and A. G. White, Grover's Search Algorithm : An optical approach, J. Mod. Optics 47, 257 (1999).

[Landauer61] R. Landauer, "Irreversibility and Heat Generation in the Computational Process", *IBM Journal of Research and Development*, 5, 1961, pp. 183-191.
[Lee99] J. Lee, Y. Kim, S. Lee, " A practical method of constructing quantum combinational logic circuits", Los Alamos physics preprint archive, quant-ph/9911053.
[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.
[Lee76] S.C. Lee, "Digital Circuits and Logic Design," Prentice Hall, Englewood Cliffs, New Jersey, 1976.
[Lee86] Lee, M. Kaveh, M. , "Fast Hadamard transform based on a simple matrix factorization", IEEE Transactions on Acoustics, Speech, and Signal Processing, Dec 1986, Volume: 34, Issue: 6, pp. 1666- 1667, ISSN: 0096-3518.
[Leibfried03] D. Leibfried, R. Blatt, C. Monroe, D.Wineland, "Quantum dynamics of single trapped ions", Review of Modern Physics, volume 75, 281 (2003).
[Lewandowski94] G. Lewandowski, "Practical Implementation and Applications Of Graph Coloring", PhD thesis, University of Wisconsin-Madison, August 1994.
[Li06] L. Li, M.A. Thornton, M.A. Perkowski, "A Quantum CAD Accelerator Based on Grover's Algorithm for Finding the Minimum Fixed Polarity Reed-Muller Form," *Proc. ISMVL 2006,* pp. 33.
[Lomont03]Ch. Lomont, "Quantum Circuit Identities," arXiv:quant-ph/0307111v1 16 July 2003.
[Luger02] G. F. Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, (San Francisco, CA: Addison-Wesley / Pearson Education Limited, 2002).
[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, S. 2003.
[Lukac05]M. Lukac, M., Perkowski, M., "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] Lukac, M., Perkowski, M., "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.

[Lukac07] Lukac, M., Perkowski, M., "Quantum mechanical model of emotional robot behaviors," in *Proceedings of the ISMVL 2007*, 2007.
[Lukac07a] Lukac , M. and M. Perkowski, "Quantum mechanical model of emotional robot behaviors," in *Proceedings of the ISMVL 2007*, 2007.

[Lukac07b] Lukac, M., Giesecke, N, Hossain, S. Kim, D.H., Perkowski, " Quantum Behaviors: Synthesis and Measurement", Proceedings of the Reed-Muller Conference, 2007, Oslo Norway.
[Lukac08] Lukac, M., PhD Thesis, 2009, Portland State University, USA.
[Manin99] Y. Manin, "Quantum computing and complexity", lecture, April 20, 1999, Johns Hopkins University, Baltimore, MD.
[Margolus03] N. Margolus, Universal cellular automata based on the collision of soft spheres. New construction in Cellular Automata, Oxford Press, 2003.

[Margolus87] N. Margolus, Physics and Computation, MIT PhD Thesis (1987). Reprinted as Tech. Rep. MIT/LCS/TR415, MIT Lab. for Computer Science, Cambridge MA.

[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.

[Maslov04] D. Maslov and G. W. Dueck. "Reversible cascades with minimal garbage," IEEE Transactions on CAD, 23(11): pp.497-1509, November 2004.
[Maslov04a] D. Maslov, N. Scott, and G. W. Dueck. (2004, Aug.) Reversible logic synthesis benchmarks page. [Online]. Available: http://www.cs.uvic.ca/~dmaslov/

[Maslov05] D. Maslov, G.W. Dueck, and D.M. Miller, "Synthesis of Fredkin–Toffoli Reversible Networks, IEEE Trans. On Very Large Scale Integration, Vol. 13, No. 6, June 2005, pp. 765-769.

[Maslov05a] D. Maslov, G.W. Dueck, D.M. Miller, "Tofffoli Network Synthesis with Templates," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005.

[Maslov05b] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. "Quantum circuit simplification using templates," in Proc. *DATE*, Munich, Germany, pp. 1208-1213, March 2005 .
[Maslov06]D. Maslov, G.W. Dueck, and D.M. Miller, "Fredkin/Toffoli Templates for Reversible Logic,"

[MATLAB] http://www.mathworks.com/
[McHugh05] D. McHugh and J. Twamley, "Trapped-ion qutrit spin molecule quantum computer," http://arxiv.org/abs/quant-ph/0506031

[McHugh05a] D. McHugh and J. Twamley, "Quantum Computer Using a Trapped-ion Spin Molecule and Microwave Radiation," *Phys. Rev. A* 71, 012315, 2005.

[McCluskey97] E. McCluskey and Ch-W. Tseng, Stuck-Fault Tests vs. Actual Defects, 1997.

[McKenzie93] L. McKenzie, A. E. A. Almaini, J. F. Miller, and P. Thompson, "Optimization of Reed-Muller Logic Functions", International Journal of Electronics, 75 (3) (1993), pp. 451-466.

[Merkle93] R. C. Merkle. "Reversible electronic logic using switches," Nanotechnology, 4:21-40, 1993.

[MCNC91] MCNC, Benchmark Functions, ftp://mcnc.mcnc.org/, MCNC, 1991

[Meyer99]D.A. Meyer, "Quantum strategies", Physical Review Letters 82(Feb. 1):1052,1999

[Miller02] D.M. Miller, Spectral and Two-Place Decomposition Techniques in Reversible Logic, IEEE Midwest Symposium on Circuits and Systems, proceedings on CD-ROM, Tulsa, OK, August, 2002.

[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.

[Miller03] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in Proc. *DAC*, Anaheim, CA, p. 318, June 2-6, 2003.

[Miller04] D. M. Miller, G. W. Dueck, and D. Maslov, "A Synthesis Method for MVL Reversible Logic", *Proc. 34th Int. Symp. On Multiple-Valued Logic*, Toronto, Canada, 19-22 May 2004, pp. 74-80.

[Miller04a]D. M. Miller, D. Maslov, and G. W. Dueck, "Synthesis of Quantum Multiple-Valued Circuits", submitted to *Journal of Multiple-Valued Logic and Soft Computing*.

[Miller05] D. Miller, and E. Fredkin, Two- state, Reversible, Universal Cellular Automata in Three Dimensions. Proceedings of the 2nd Conference on Computing Frontiers, Ischica, Italy, pp. 45—51, 2005.

[Miller05] D. Maslov, G. W. Dueck, and N. Scott, "Reversible Logic Synthesis Benchmarks," [Online document], Available HTTP: http://www.cs.uvic.ca/~dmaslov/, November 15, 2005 [cited December 5, 2006].

[Miller06] D. M. Miller, D. Maslov, and G. W. Dueck, "Synthesis of Quantum Multiple-Valued Circuits", *J. MVL.*, Vol. 12, No. 5-6, 2006.

[Miller94a] J. F. Miller and P. Thomson, "A Highly Efficient Exhaustive Search Algorithm for Optimizing Canonical Reed-Muller Expansions of Boolean Functions", Int. J. of Electron., 76 (1994), pp. 37-56.

[Miller94b] J. F. Miller, H. Luchian, P.V.G. Bradbeer, and P.J. Barclay, "Using a Genetic Algorithm for Optimizing Fixed Polarity Reed-Muller Expansions of Boolean Functions", Int. J. Electron., 76 (1994), pp. 601-609.
[Miller97] J. F. Miller, P. Thomson, and T. Fogarty, "Chapter 6 - Designing Electronic Circuits using Evolutionary Algorithms. Arithmetic Circuits: A Case Study", in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications. Editors: D. Quagliarella, J. Periaux, C. Poloni, and G. Winter. New York: John Wiley & Sons, Inc., 1997.
[Mintert01] F. Mintert and C. Wunderlich, Ion-Trap Quantum Logic Using Long-Wavelength Radiation, *Phys. Rev. Lett.* 87, 257904, 2001. Also quant-ph/0104041
[Mishchenko01] A. Mishchenko and M. Perkowski, "Fast heuristic minimization of exclusive-sums-of-products", *Proc. Reed-Muller Workshop 2001*, 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.
[Mizel07] Mizel A., Lidar D., Mitchell M.: *Simple proof of equivalence between adiabatic quantum computation and the circuit model.* APS March Meeting, Denver, Colorado, 2007. http://www.aps.org/meeting/march
[Moore65] G. E. Moore, (1965) Electronics 38:8
[Morita92] K. Morita, S. Ueno, Computation—Universal Models of Two—Dimensional 16-state Reversible Cellular Automata, IEICE Trans. Inf. & Syst , E75-D, 1, pp.141—147, 1992.
[Morita94] K. Morita. Reversible cellular automata. J. Information Processing Society of Japan, 35: 315—321, 1994.
[Muller54] D.E. Muller, "Applications of Boolean Algebra to Switching Circuit Design and to Error Detection," *IRE Trans. Electronic Computers,* vol. 3, pp. 6-12, 1954.
[Muthurkrishnan00] 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.
[Negotevic02] G. Negotevic, M. Perkowski, M. Lukac, and A. Buller, "Evolving quantum circuits and an FPGA based quantum computing emulator", Proc. Fifth Intern. Workshop on Boolean Problems, September 19-20 2002, Freiberg, Sachsen, Germany.

[Newton86] A.R. Newton, and A.L.Sangiovanni-Vincentelli, "Computer-Aidded Design for VLSI Circuit," IEEE Computer, Apr. 1986.
[VonNeumann66] J.von Neumann. The Theory of Self-Reproducing Automata. A. W. Burks (ed.), University of Illinois Press, Urbana, IL, 1966.

[Nguyen87] L. Nguyen, M. Perkowski, N. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers", Proceedings of 24th Design Automation Conference, June 28 - July 1, 1987, Miami, Florida, Paper 33.3.
[Nielsen97] M. A. Nielsen and Isaac L. Chuang , "Programmable Quantum Gate Arrays", Phys. Rev. Lett. 79, 321 - 324 (1997).

[Nielsen98] MA Nielsen, E Knill, R Laflamme, "Complete quantum teleportation using nuclear magnetic resonance", Arxiv preprint quant-ph/9811020, 1998 - arxiv.org

[Nielsen00] M. Nielsen and I. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.

[Nielsen02] M. A. Nielsen, M. J. Bremner, J. L. Dodd, A. M. Childs and C. M. Dawson, "Universal simulation of Hamiltonian dynamics for quantum systems with finite-dimensional state spaces," *Phys. Rev. A* 66, 022317, 2002

[Nilsson65] N. J. Nilsson, Learning Machines, McGraw-Hill, New York, 1965.

[Nilsson71] N.J. Nilsson, "Problem-Solving Methods in Artificial Intelligence," Mc Gram Hill, New York, 1971.

[Nilsson98] N. J. Nilsson, Artificial Intelligence: A New Synthesis, (San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1998).

[Pakula06] I. Pakula, E.W. Piotrowski, J. Sladkowski, "Quantum market games: implementing tactics measurements" Journal of Physics: Conference Series 30 (2006) pp. 56-59.

[Pai96] Pai D.K., Barman R., *Constraint Programming for Platonic Beast Legged Robots.* In: Proc. Intern. Conf. on Robotics and Automation, Minneapolis, 1996.

[Paul90] W. Paul ,"Electromagnetic traps for charged and neutral particles", Rev. Mod. Phys, 62, 531,(1990).

[Peres85] A. Peres, "Reversible Logic and Quantum Computers," *Physical Review A*, 32:3266-3276, 1985.

[Peres00] A. Peres, H. Bechmann-Pasquinucci, "Quantum Cryptography with 3-State Systems", *Phys. Rev. Lett.* 85, 3313, 2000.

[Perkowski78] M. Perkowski, "The state-space approach to the design of multipurpose problem-solver for logic design," Proceedings of the IFIP WG.5.2 Working Conference "Artificial Intelligence and Pattern Recognition in Computer-Aided Design", Grenoble, France, 17-19 March 1978, J. C. Latombe (ed.) North Holland, Amsterdam, pp. 124-140, 1978.

[Perkowski82] M. Perkowski, "Digital Devices Design by Problem-Solving Transformations," Journal on Computers and Artificial Intelligence (Pocitace a umela intelligencia), Vol. 1, No. 4, August 1982, pp. 343-365.

[Perkowski89] M. Perkowski, M. Helliwell, and P. Wu, "Minimization of Multiple-Valued Input Multi-Output Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions", Proc. of the 19th International Symposium on Multiple-Valued Logic, May 1989, pp. 256-263.

[Perkowski91] M. Perkowski and P. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms", Proc. of the 3rd NASA Symposium on VLSI Design, University of Idaho, Oct. 30-31, 1991, pp. 11.3.1 – 11.3.13.

[Perkowski92] M. Perkowski, "Generalized Orthonormal Expansion and Some of Its Applications", Proc. of the International Symposium On Multiple-Valued Logic, Sendai, Japan, May 1992, pp. 442-450.

[Perkowski92a] M. Perkowski, L. Csanky, A. Sarabi, and I. Schafer, "Minimization of Mixed-Polarity AND/XOR Forms", Proc. of the IEEE International Conference on Computer Design, ICCD'92, October 11-13, 1992, Boston, Massachusetts, pp. 32-36, 1992.

[Perkowski95] M. Perkowski, A. Sarabi, and F.R. Beyl, "Universal XOR Canonical Forms of Boolean Function and its Subset Family of AND/OR? XOR Canonical Forms", IEEE workshop on Logic Synthesis, 1995.

[Perkowski95a] M. A. Perkowski, T. Ross, D. Gadd, J. A. Goldman, and N. Song, "Application of ESOP Minimization in Machine Learning and Knowledge Discovery", Proc. of the Second Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Chiba City, Japan, August 27-29, 1995, pp. 102-109.

[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).

[Perkowski97a] M. Perkowski, L. Jozwiak, and R. Drechsler, "A Canonical AND/EXOR Form that Includes both the Generalized Reed-Muller Forms and Kronecker Reed-Muller Forms", Proc. of the Reed-Muller 1997 Conference, Oxford University, U.K., Sept. 1997, pp. 219-233.

[Perkowski97b] M. Perkowski, L. Jozwiak, R. Drechsler, and B. Falkowski, "Ordered and Shared, Linearly Independent, Variable-Pair Decision Diagrams", Proc. of the First International Conference on Information, Communications and Signal Processing, ICICS'97, Singapore, September 9-12, 1997, Session 1C1: Spectral Techniques and Decision Diagrams.

[Perkowski97c] M. Perkowski, L. Jozwiak, and R. Drechsler, "New Hierarchies of AND/EXOR Trees, Decision Diagrams, Lattice Diagrams, Canonical Forms, and Regular Layouts", Proc. of the Reed-Muller 1997 Conference, Oxford Univ., U.K., Sept. 1997, pp. 115 - 132.

[Perkowski97d] 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).

[Perkowski99] M. Perkowski, R. Malvi, S. Grygiel, M. Burns, and A. Mishchenko, "Graph Coloring Algorithms for Fast Evaluation of Curtis Decomposition", Proc. of the Design Automation Conference, (DAC'99), June 21-23, 1999.

[Perkowski99a] M. Perkowski, U. Kalay, D. Hall, and A. Shahjahan, "Rectangular Covering Factorization of ESOPs into Scan-Based Levelized Circuits with Universal Test Set", Proc. of Reed-Muller '99, University of Victoria, Victoria, B.C., Canada, August 20-21, 1999.

[Perkowski99b] M. Perkowski, U. Kalay, D. Hall, and A. Shahjahan, "Rectangular Covering Factorization of ESOPs into Scan-Based Levelized Circuits with Universal Test Set", Proc. of Reed-Muller '99, University of Victoria, Victoria, B.C., Canada, August 20-21, 1999.

[Perkowski99c] M. Perkowski, A. Chebotarev, and A. Mishchenko, "Evolvable Hardware or Learning Hardware? Induction of State Machines from Temporal Logic Constraints", Proc. of the First NASA/DOD Workshop on Evolvable Hardware, Jet Propulsion Laboratory, Pasadena, California, July 19-21, 1999.

[Perkowski99d] 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.

[Perkowski99e] 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.

[Perkowski01] M. Perkowski, et al, "Regularity and Symmetry as a base for efficient Realization of Reversible Logic Circuits", *Proc. Intel Workshop on Logic Synthesis,* June 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. 201-209

[Perkowski04] M. Perkowski, "From Quantum Gates to Quantum Learning: recent research and open problems in quantum circuits", invited paper, Proceedings of 6th International Workshop on Boolean Problems, Freiberg University of Mining and Technology, Germany, September 23-24, 2004, pp. 1-16.

[Perkowski05] M. Perkowski, "Multiple-Valued Quantum Circuits and Research Challenges for Logic Design and Computational Intelligence Communities," *Invited Paper, IEEE ConneCtIonS,* IEEE Computer Intelligence Society, November 2005, pp. 6-12.

[Perkowski05a] Perkowski M., Sasao T, Kim J-H., Lukac M., Allen J., Gebauer S.: *Hahoe KAIST Robot Theatre: Learning Rules of Interactive Robot Behavior as a Multiple-Valued Logic Synthesis Problem.* In: Proc. ISMVL 2005, pp. 236-248.
[Perkowski07] M. Perkowski, J. Biamonte and M. Lukac, "Test Generation and Fault Localization for Quantum Circuits," *Proceedings of the 35th International Symposium on Multiple-Valued Logic,* May 19-May 21, 2005 at Calgary, Canada.
[Perkowski07a] M. Perkowski, *Quantum Robotics for Teenagers*, book in preparation. 2007.

[Perkowski07b] Perkowski M.: *Quantum Algorithms for Robot Vision*. Report, PSU Intelligent Robotics Laboratory, 2007.
[Perus96] M. Perus, "Neuro-Quantum Parallelism in Brain-Mind and Computers", Informatica, v20, pp. 173-183, 1996.

[Pierce05] D. Pierce, J. Biamonte and M. Perkowski, "Test Set Generation and Fault Localization Software for Reversible Circuits," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005.
[Pojak95] S. Pojak , Z. Tuza, " Maximum cuts and largest bipartite subgraphs", DIMACS:Series in Descrete Mathematics and Theoretical Computer Science, 1995, Volume 20.

[Pradhan87] D. K. Pradhan, Fault-Tolerant Computing: Theory and Techniques, Vol. 1, (Upper Saddle River, New Jersey: Prentice-Hall, 1987).

[Pradhan78] D. K. Pradhan, "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays", IEEE Trans. On Comp., Vol. 27, No. 2, pp. 181-187, Feb. 1978.

[Price99] M.D. Price, S.S. Somaroo, C.H. Tseng, J.C. Core, A.H. Fahmy, T.F. Havel and D.Cory, "Construction and Implementation of NMR Quantum Logic Gates for Two Spin Systems," *Journal of Magnetic Resonance*, 140, pp. 371-378, 1999

[Price99a]M.D. Price, S.S. Somaroo, A.E. Dunlop, T.F. Havel, and D.G. Cory, "Generalized methods for the development of quantum logic gates for an NMR quantum information processor," *Physical Review A*, Vol. 60, No. 4, October 1999, pp. 2777-2780.

[Quinlan93]J. R. Quinlan, C4.5: Programs for Machine Learning, (Palo Alto, California: Morgan Kaufmann, 1993).
[QuIDDPro] QuIDDPro: High-Performance Quantum Circuit Simulation, http://vlsicad.eecs.umich.edu/Quantum/qp/
[Raghuvanshi07] A. Raghuvanshi, Y. Fan, M. Woyke, A. Kumar, and M. Perkowski, "Quantum robots for teenagers," in *Proceedings of the ISMVL 2007*, 2007.
[Raussendorf01] R. Raussendorf and H. J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.,* 86:5188-5191, 2001.

[Reddy72] S. M. Reddy, "Easily Testable Realizations for Logic Functions", IEEE Trans. On Comp., Vol. 21 (1972), pp. 1183-1188.
[Reed54] I.S. Reed. A class of multiple-error-correcting codes and their decoding scheme. *IRE Trans. Of Inf. Theory,* 3:6-12, 1954.
[Riege92] M. W. Riege, P. W. Besslich, "Low-Complexity Synthesis of Incompletely Specified Multiple-Output Mod-2 Sums", IEE Proc. E., 139 (4) (1992), pp. 355-362.
[Rudell85] R. L. Rudell and A. L. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for Multiple-Valued Logic Minimization", Proc. of the IEEE Custom Integrated Circuits Conference, 1985
[Rubinstein01] B.I.P. Rubinstein, "Evolving quantum circuits using genetic programming", *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001),* pp. 144-151, 2001.
[Salmon89] J. V. Salmon, E. P. Pitty, and M. S. Abramson, "Syntactic Translation and Logic Synthesis in Gatemap", IEE Proceedings, Vol. 136, Part E, No. 4, July 1989, pp. 321-328.

[Samuel59] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," IBM J., Vol. 3., 1959, pp. 210-129.

[Samuel67] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers II, Recent Progress", IBM Journal of Research and Development, Vol. 11, (1967), No. 6, pp. 601-617.

[Sarabi92] A. Sarabi and M. A. Perkowski, "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks", Proc. 1992 IEEE Design Automation Conference, June 1992, pp. 30-35

[Sarabi93] A. Sarabi and M. Perkowski, "Design for Testability Properties of AND/XOR Networks", IFIP WG10.5, Proc. of the Workshop on Applications of the Reed-Muller Expansion in Circuit Design 1993, pp. 147-153.

[Sasao78] T. Sasao, "An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays", Proc. of the 8th International Symposium on Multiple-Valued Logic (ISMVL'78), May 1978, pp. 65-72.

[Sasao91] T. Sasao, "A Transformation of Multiple-Valued Input To-Valued Output Functions and its Application to Simplification of Exclusive-Or Sum-of-Products Expressions", Proc. of the International Symposium of Multi-Valued Logic 1991 (ISMVL-91), May 1991, pp. 270-279.

[Sasao90] T. Sasao and P. Besslich, "On the Complexity of MOD-2 Sum PLAs", IEEE Transactions on Computers, Vol. 39, No. 2, (February 1990), pp. 262-266.

[Sasao91a] T. Sasao, "On the Complexity of Some Classes of AND-EXOR Expressions", IEICE Technical Report FTS 91-35, October 1991.

[Sasao81]T. Sasao, "Multiple-Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays", IEEE Transactions on Computing, Vol. C-30, September 1981, pp. 635-643.

[Sasao86] T. Sasao, "MACDAS: Multi-level AND-OR Circuit Synthesis using Two-Variable Generators", Proc. of the 23rd Design Automation Conference, Las Vegas, June 1986, pp. 86-93.

[Sasao90a] T. Sasao, "EXMIN: A Simplification Algorithm for Exclusive-Or-Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions", Proc. of the International Symposium on Multi-Valued Logic, (ISMVL-90), May 1990, pp. 128-135.

[Sasao90c] T. Sasao and P. Besslich, "On the Complexity of MOD-2 Sum PLAs", IEEE Transactions on Computers, Vol. 39, No. 2, (February 1990), pp. 262-266.

[Sasao91d] T. Sasao, "A Transformation of Multiple-Valued Input To-Valued Output Functions and its Application to Simplification of Exclusive-Or Sum-of-Products Expressions", Proc. of the International Symposium of Multi-Valued Logic 1991 (ISMVL-91), May 1991, pp. 270-279.

[Sasao91e] T. Sasao, "On the Complexity of Some Classes of AND-EXOR Expressions", IEICE Technical Report FTS 91-35, October 1991.

[Sasao93] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR-Sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Trans.on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, No. 5, May 1993, pp. 621-632.

[Sasao93e] T. Sasao, Logic Synthesis and Optimization, (Norwell, Massachusetts: Kluwer Academic Publishers, 1993)

[Sasao94] T. Sasao and D. Debnath, "An Exact Minimization Algorithm for Generalized Reed-Muller Expressions", Proc. of the IEEE Asia-Pacific Conference on Circuits and Systems, (APCCAS'94), Taipei, Taiwan, pp. 460-465, Dec. 1994.

[Sasao95f] T. Sasao and M. Perkowski, EXOR Logic Synthesis (Boston: Kluwer Academic Publishers, 1995), pp. 19-32.

[Sasao95g] T. Sasao, "Easily Testable Realizations for Generalized Reed-Muller Expressions", IEEE Trans. Comp., Vol. 46, No. 6, pp. 709-716, 1997.

[Sasao96] T. Sasao, M. Fujita, Representations of Discrete Functions, *Kluwer Academic Publishers*, 1996

[Saul92] J. Saul, "Logic Synthesis for Arithmetic Circuits using the Reed-Muller Representation", Proc. of the European Conf. on Design Automation 1992, pp. 109-113.

[Schaefer91] I. Schaefer and M. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms", Proc. of the International Symposium on Multi-Valued Logic, (ISMVL'91), May 1991, pp. 40-48.

[Schaefer93] I. Schaefer and M. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits fo Incompletely Specified Multi-Output Boolean Functions with Mapping

Multiplexer Based FPGAs", IEEE Transactions on Computer Aided Design, Vol. 12, No. 11, November 1993, pp. 1655-1664.

[Schrödinger26] E. Schrödinger, "Quantisierung als Eigenwertproblem", Annalen der Physik 79, 361(1926).

[Scully01] M.O.Scully and M.S. Zubairy, Quantum Optical Implementation of Grover's Algorithm, Proc. Natl. Acad. Sci. USA 98,9490(2001).

[Shah09] Shah, D., "Synthesis of Binary and Multi-valued Quantum Circuits with Regular Structures", PhD Thesis, preparation, 2009, Portland State University, USA.

[Shannon49] C. E. Shannon, "The Synthesis of Two-Terminal Switching Circuits", Bell System Technical Journal, Vol. 28, pp. 59-98, January 1949.

[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.

[Shende02a] V.V. Shende, A.K. Prasad, I.L. Markov, J.P. Hayes, "Reversible Logic Circuit Synthesis," *Proc. 11th IEEE/ACM Intern. Workshop on Logic Synthesis (IWLS),* 2002, pp. 125 – 130.

[Shende03] V.V. Shende, A.K. Prasad, I.L. Markov and J.P. Hayes, Synthesis of Reversible Logic Circuits, IEEE Trans. on CAD 22, 710 (2003).

[Shende03a] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," IEEE Trans. Computer-Aided Design Integr. Syst., vol. 22, no. 6, pp. 723–729, Jun. 2003.

[Shende04] Shende et al new 2004 and 2005 matrix decomposition papers.

[Shivgand05] V.S. Shivgand, A.Aulakh, and M. Perkowski, "Quantum Circuit Layout," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005.

[Shor94] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring" in Proc. 35th Annu. Symp. On the Foundations of Computer Science (ed. Goldwasser, S.) 124-134 (IEEE Computer Society Press, Los Alamitos, California, 1994).

[Shor97] P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26, 1484-1509 (1997).

[Sipper97] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems", IEEE Transactions on Evolutionary Computation, April 1997, Vol. 1, No. 1, pp. 83-97.

[Smolin96] J. Smolin, D. P. DiVincenzo, "Five two-qubit gates are sufficient to implement the quantum Fredkin gate." Physical Review A, Vol. 53, no. 4, April 1996, pp. 2855-2856.

[Song05] X. Song, G. Yang, and M. Perkowski, ``Algebraic Characteristics of Reversible Gates," Accepted to *Theory of Computing Systems (Mathematical Systems Theory),* Springer Verlag. First published on Online Test, ISSN 1432-4350. 2005.
[Slagle70] J. It. Slagle, "Artificial Intelligence: the Heuristic Programming Approach," McGraw Hill, New York 1970.

[Simon97] D. Simon, "On the Power of Quantum Computation", SIAM Journal of Computation, v26, pp. 1474-83, 1997.

[Software1] Software may be run at PSU from the directory: /stash/polo/benchmarks/MCNC.

[Software2] Portland State University. Electrical Engineering Computer System. Online Manuals, /stash/polo/man/man1, files: cgrmin.1, espresso.1, and exorcism.1. Information regarding CGRMIN, ESPRESSO, and EXORCISM software.

[Song93] N. Song, "Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions", Master of Science Thesis, Portland State University, 1993.

[Song06] Song X., Yang G., Perkowski M.: *Algebraic Characteristics of Reversible Gates.* Theory of Computing Systems (Mathematical Systems Theory), Springer Verlag. *39(2),* 2006, pp. 311-319.
[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.

[Stankovic97]R. Stankovic and R. Drechsler, "Circuit Design from Kronecker Galois Field Decision Diagrams for Multiple-Valued Functions", 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).

[Steane97] A. Steane, "The ion trap quantum information processor", Appl. Phys. B. 64, 623 (1997).

[Stedman05] Ch. Stedman, B. Yen and M. Perkowski, "Synthesis of Reversible Circuits with Small Ancilla Bits for Large Irreversible Incompletely Specified Multi-Output Boolean Functions" *Proceedings of the 14th International Workshop on Post-Binary ULSI Systems,* May 18, 2005, Calgary, Canada.
[Stewart89] I. Stewart, Galois Theory, 2nd Edition, (New York: Chapman & Hall, 1989).

[Styer02] D. F. Styer *et al.*, "Nine formulations of quantum mechanics", American Journal of Physics 70, 288 (2002).

[Svore04] K. Svore, T.G. Draper, S.A. Kutin, and E.M. Rains, "Logarithmic-depth Quantum Carry-lookahead Adder, *quant-ph/04061 42.*

[Thompson03] A. Thompson, I. Harvey, and P. Husbands, "Unconstrained Evolution and Hard Consequences", School of Cognitive and Computing Sciences, University of Sussex, Brighton BN1 9QH, UK (Electronically available from: http://www.cogs.susx.ad.uk/users/adrianth.).

[Thompson95] A. Thompson, "Evolving Electronic Robot Controllers that Exploit Hardware Resources", Advances in Artificial Life: Proc. of the Third European Conference on Artificial Life, Granada, Spain, June 4-6, 1995, F. Moran, A. Moreno, J. J. Merelo, and P. Chacon, Editors, (Berlin, Germany: Springer-Verlag, 1995.)

[Thompson95a] A. Thompson, "Evolving Fault Tolerant Systems", First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, (GALESIA'95), Sheffield, September.

[Toffoli90] T. Toffoli and N. Margolus. Invertible cellular automata: A review. Physica D, 45:229—253, 1990.

[Thornton05] M. Thornton "The Karhunen-Loève Transform of Discrete MVL Functions," *IEEE International Symposium on Multiple-Valued Logic* (ISMVL), May 18-21, 2005.

[Tran89] A. Tran, "Tri-state map for the minimisation of exclusive-OR switching functions", IEE Proceedings on Computers and Digital Techniques, - Jan 1989, Volume: 136, Issue: 1, pp.16- 21.

[Tsai96] C. Tsai and M. Marek-Sadowska, "Multilevel Logic Synthesis for Arithmetic Functions", Proc. of the 33rd Design Automation Conference (DAC), Las Vegas, NV, 1996.

[Ventura98] D. Ventura, T. Martinez ,"Quantum Associative Memory", IEEE Transactions on Neural Networks, 1998.

[Ventura99] D. Ventura, T. Martinez ,"Initializing the Amplitude Distribution of a Quantum State", Foundations of Physics Letters, 1999 – Springer, pp. 547-559.

[Viamontes04] G. F. Viamontes, I. L. Markov and J. P. Hayes, "Improving Gate-Level Simulation of Quantum Circuits" (quant-ph/0309060), to appear in *Quantum Information Processing*, 2004.

[Varma91] D. Varma and E. A. Trachtenberg, "Computation of Reed-Muller Expansions of Incompletely Specified Boolean Functions from Reduced Representations", IEE Proc. E., 138 (2) (1991), pp. 85-92.

[Vedral96] V. Vedral, A. Barenco, and A. Ekert, "Quantum Networks for elementary arithmetic operations," *Phys. Rev. A.* 54:147, 1996

[Waltz75] Waltz D.L, "*Understanding Line Drawings of Scenes with Shadows*", In: P. H. Winston ed. Psychology of Computer Vision. McGraw-Hill, N.Y., 1975. pp.19-91.

[Weiss01] P.S Weiss, J. Tersoff, A. M. Chang, K. Likharev, J. Van, "Programmable and autonomous computing machine made of biomolecules", Nature, nature.com, 2001.

[Williams99] C.W. Williams, A.G. Gray, "Automated Design of Quantum Circuits", ETC Quantum Computing and Quantum Communication, *QCQC 1998,* Palm Springs, California, February 17-20, *Springer-Verlag*, pp. 113-125, 1999.

[Winter74] D. J. Winter, The Structure of Fields, (Berlin, Germany: Springer-Verlag, 1974).

[Wineland94] Wineland DJ, Bollinger JJ, Itano WM, Moore FL (1992) Phys Rev A 46:R6797; Wineland DJ, Bollinger JJ, Itano WM (1994) Phys Rev A 50:67
[Wineland98] D. J. Wineland, C. Monroe, W. M. Itano, D. Leibfried, B. E. King, and D. M. Meekhof, "Experimental Issues in Coherent Quantum-State Manipulation of Trapped Atomic Ions", Journal of Research of the National Institute of Standards and Technology 103, 259 (1998).

[Wireworld1] http://mathworld.wolfram.com/WireWorld.html

[Wolfram02] S. Wolfram. A new kind of Science. Wolfram Media, 2002.

[Wong89]Wong A.K.C., Lu S.W., Rioux M.: *Recognition and shape synthesis of 3D objects based on attributed hypergraphs.* IEEE Trans. on Pattern Anal. and Mach. Intel., 1989. 11. pp. 279-290.
[Wu96] H. Wu, M. Perkowski, X. Zeng, and N. Zhuang, "Generalized Partially-Mixed-Polarity Reed-Muller Expansion and Its Fast Computation", IEEE Transactions on Computers, Vol. 45, No. 9, September 1996, pp. 1084-1088.

[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.
[Yang04a] G. Yang, W. Hung, X. Song, and M. Perkowski, "Depth Limited Realization of Reversible Logic," *Microelectronics,* March 2004.
[Yang04b] G. Yang, X. Song, and M. Perkowski, "Minimal Universal Library," *Discrete Applied Mathematics,* 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.

[Yang05a] G. Yang, X. Song, W. N. N. Hung, and M. Perkowski, "On Realization of 3-qubit Reversible Circuits with the minimum number of non-linear gates", 7th International Symposium on Representations and Methodology of Future Computing Technologies (RM2005), Tokyo, Japan, September 2005.

[Yang05b] G. Yang, W. Hung, X. Song, and M. Perkowski, "Majority-Based Reversible Logic Gates", *Theoretical Computer Science* C. 334(1-3), pp. 259-274, April 15, 2005. ISSN 0304-3975.

[Yang05b] G. Yang, X. Song, M. Perkowski, Fast Synthesis of Exact Minimal Reversible Circuits using Group Theory, *ACM/IEEE ASP-DAC (Asia and South Pacific Design Automation Conference),* Shanghai, People's Republic of China, January 2005.
[Yang05c] G. Yang, X. Song, M. Perkowski, "Bi-direction synthesis for reversible circuits," *Proc. IEEE Computer Society Annual Symposium on VLSI,* 2005.
[Yang05d] G. Yang, X. Song, M. Perkowski, and W.N.N. Hung, "Minimal Universal Library for n*n Reversible Circuits," *Proc. International Symposium on Representations and Methodologies for Emergent Computing Technologies,* Tokyo, Japan, September 2005.
[Yang05e] G. Yang, X. Song, M. Perkowski, and J. Wu, "Realizing ternary quantum switching networks without ancilla bits," *Journal of Physics A. Mathematical and General,* 2005.
[Yang05f] G. Yang, X. Song, W. Hung and M. Perkowski, "Bi-directional synthesis for reversible circuits," *IEEE Transactions on VLSI Systems,* 2005
[Yang05g] G. Yang, X. Song, and M. Perkowski, "On realization of 3-qubit reversible circuits," *Journal of Circuits, Systems, and Computers (JCSC),* World Scientific Publishers, 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.
[Yang06] G. Yang, F. Xie, X. Song and M.A. Perkowski, "Universality of two-qudit ternary reversible gates", *J. Phys. A: Math. Gen.,* 2006, Vol. **39,** pp. 7763-7773.
[Yen05] B. Yen, P. Tomson, and M. Perkowski, "Sum of Non-disjoint Cubes Covering Generation for Multi-Valued Systems of base 2, for use in Muthukrishnan-Stroud Quantum Realizable Gates: An Extension of the EXOR Covering Problem", *Proceedings of* IWLS 2005. June 2005.
[Younes03] A. Younes, J. Miller, "Automated method for building CNOT based quantum circuits for Boolean Functions", Technical report CSR-03-3, University of Birmingham, Los Alamos physics preprint archive, quant-ph/0304099.

[Zadeh83] L.A. Zadeh, "A Fuzzy-set-theoretic approach to the compositionality of meaning:propositions,dispositions and canonical forms", Journal of Semantics 1983 2(1):253-272

[Zadorozhny78] V.N. Zadorozhny, "Realization of Logic Functions by Arithmetical Expressions," *Automatization for Computer Structure Analysis and Synthesis,* Novosibirsk: Engineering Building Inst., Russia, 1978 (in Russian).

[Zakrevskij95] A. Zakrevskij, "Minimum Polynomial implementation of Systems of Incompletely Specified Boolean Functions", IFIP WG 10.5, Proc. of the Workshop on Applications of the Reed-Muller Expansion in Circuit Design, August 27-29, 1995, Makuhari, Chiba, Japan.

[Zalka99] Ch. Zalka, "Grover's Quantum Searching algorithm is optimal", *Phys. Rev. A* 60, pp. 2746–2751, 1999.

[Zeng95] X. Zeng, M. Perkowski, K. Dill, and A. Sarabi, "Approximate Minimization of Generalized Reed-Muller Forms", IFIP WG 10.5, Proceedings of the Workshop on Applications of Reed-Muller Expansion in Circuit Design, 27-29 August 1995, Makuhari, Chiba, Japan, pp. 221-230.

[Zhang99] W. Zhang, State Space Search. Algorithms, Complexity, Extensions, and Applications, Springer, 1999.

[Zhegalkin29] I. L. Zhegalkin, "Arithmetization of Symbolic Logic", (in Russian), Mathematiceskij Sbornik, Vol. 35, pp. 311-373, (1928), Vol. 36, pp. 205-338 (1929).

[Zhirnov03] V. V. Zhirnov, R. K. Kavin, J. A. Hutchby, and G. I. Bourianoff, "Limits to Binary Logic Switch Scaling – A Gedanken Model", *Proc. of the IEEE*, 91, no. 11, 2003, pp. 1934-1939.

[Zilic93] Z. Zilic, Z. G. Vranesic, "Multiple-Valued Logic in FPGAs", Proc. of the Midwest Symposium on Circuits and Systems, Vol. 2, pp. 1553-1556, 1993.

[Zilic95] Z. Zilic and Z. Vranesic, "A Multiple-Valued Reed-Muller Transform for Incompletely Specified Functions", IEEE Trans. On Comput., 44 (8) (1995), pp. 1012-1020.

[Zilic93a] Z. Zilic, Z. G. Vranesic, "Current-mode CMOS Galois Field circuits", Proc. of the International Symposium on Multi-Valued Logic 1993, (ISMVL'93), pp. 245-250, 1993.

[Zilic02] Z. Zilic and K. Radecka, "The Role of Super-Fast Orthogonal Transforms in Speeding up Quantum Computations," ISMVL 2002.

END.