A Quantum CAD Accelerator Based on Grover's Algorithm for Finding the Minimum Fixed Polarity Reed-Muller Form

Lun Li[1]*, Mitch Thornton*, and Marek Perkowski**

***CAD Methods Lab**, *Southern Methodist University, Dallas, TX 75206, USA*, {lli,mitch}@engr.smu.edu
**** Portland Quantum Logic Group,** *Portland State University, Portland, OR 97207, USA, mperkows@ee.pdx.edu*

Abstract

We describe the use of Grover's algorithm as implemented in a quantum logic circuit that produces a solution for a classical switching circuit design problem. The particular application described here is to determine a Fixed Polarity Reed-Muller (FPRM) form that satisfies a threshold value constraint, thus we find a particular FPRM form among all 2^n FPRM forms that has a number of terms less than or equal to the threshold value. Grover's algorithm is implemented in a quantum logic circuit that also contains a subcircuit that expresses all possible FPRM solutions of a given function. This approach illustrates how fast transforms as known from spectral theory can be combined with quantum computing as a part of an oracle.

1. Introduction

The original quantum search algorithm of Grover finds a single solution, a number 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 with value "1" (a true minterm) in a Karnaugh Map of a binary-valued function F in which all other cells have values 0. Obviously, if no additional information is available, classical *SAT* algorithms can be employed that have worst-case exponential behavior. When there are $M>1$ solutions, a variant of Grover's algorithm can be employed to find all solutions (*SAT-ALL*).

A generic machine-learning problem can be formulated as finding the simplest rule describing a Boolean function representing the set of all solutions. In this work, the rule is expressed as a Fixed Polarity Reed-Muller expression; however, variations for other types of expressions such as Exclusive-OR Sum of Products (ESOP) are also possible. In the case of a single solution, our algorithm reduces to the classical algorithm of Grover with the "database" that is searched being that of a binary-valued vector of length 2^n. From this aspect, this research is only of theoretical value since it is unlikely that a quantum computer with 2^n qubits will be built in the near future; however, it is of interest that classical EDA problems can be sped-up using quantum computing principles.

A classic problem in binary logic minimization is that of minimizing a single-output Boolean function using a two-level structure consisting of the exclusive-OR of an array of ANDs of literals. If there is no constraint on the literals, the expression is called the Exclusive Sum of Product expression or ESOP. When all literals are not negated, the problem is that of finding the Positive Polarity Reed-Muller form (PPRM). 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 3 ($a=0,b=0$) and the equivalent PPRM $F=1\oplus a\oplus b\oplus ab$ has the polarity number 0 ($a=1,b=1$).

Several heuristic methods have been formulated in the past for both ESOP minimization [19,20] and for FPRM minimization [7,8]. In this work, we present a fundamentally new approach to FPRM minimization that is based on quantum logic and the use of Grover's algorithm. This approach can be extended to several canonical XOR forms [7] as well as to the non-canonical ESOPs; however, here 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 transform that can be described by a butterfly diagram.

2. FPRM and Quantum Logic Background

This section contains preliminary background discussion of two basic topics; the theory of FPRM forms and the basics of quantum logic circuits.

[1] This work was supported in part by the Advanced Technology Program (ATP) of Texas under grant 003613-0029-2003.

2.1. FPRM Theory

A switching function is commonly described in a sum-of-minterms form that is canonic and 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

$$F = m_0\overline{x}_1\overline{x}_2\overline{x}_3 + m_1\overline{x}_1\overline{x}_2x_3 + m_2\overline{x}_1x_2\overline{x}_3 + m_3\overline{x}_1x_2x_3$$
$$+ m_4x_1\overline{x}_2\overline{x}_3 + m_5x_1\overline{x}_2x_3 + m_6x_1x_2\overline{x}_3 + m_7x_1x_2x_3$$

where $m_i \in \{0,1\}$ are commonly referred to as the minterms of the function *f*. 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

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

where $a_i \in \{0,1\}$ are commonly referred to as the FPRM spectral coefficients and $\dot{x}_i$ represents a literal in either complemented or uncomplemented form consistent for all values of *i*. The particular assignment of polarities of dependent variables $\dot{x}_i$ leads to the polarity number. For example, $x_1x_2x_3 \rightarrow 0$, $x_1\overline{x}_2\overline{x}_3 \rightarrow 3$, $\overline{x}_1x_2x_3 \rightarrow 4$, etc.

The problem we address here is that of finding a polarity number $P \le T$ such that a Reed-Muller expansion can be formed where at most T spectral coefficients are non-zero. 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.

The two expressions shown previously are both affine functions that are related by a linear transformation. This transformation is well-known and commonly characterized by a linear transformation matrix as the fixed-polarity Reed-Muller transform [16,17,18]. 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. Here, we use the symbol ⊗ to denote the Kronecker product (Kronecker multiplication) operation. As an example, the transformation matrix for the PPRM transformation is given as

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

M_{N1} and M_{P1} are used to denote transformation matrices for complemented and positive-polarity variables respectively. As an example the transformation matrix for an FPRM of polarity 5 (101_2) 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 commonly attributed to [21] may be used to represent the transformation in a 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, as shown in Figure 1.

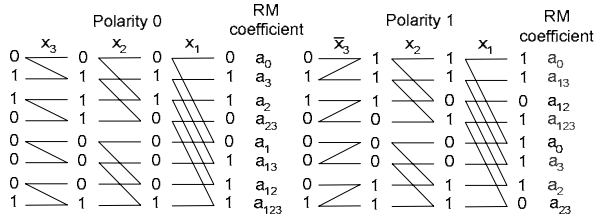


Figure 1: Butterfly signal flow-graph

As described in detail in [9] butterfly diagrams may be devised for any given polarity number for a FPRM expansion. Unfortunately, finding the "best" polarity number and corresponding maximal number of zero-valued Reed-Muller coefficients resulting in the "best" butterfly structure is very challenging. In this work, we show how the use of quantum logic circuits can give an optimal polarity number with respect to a threshold value.

2.2 Quantum Logic Circuits

One of the most important quantum gates is the quantum XOR gate (also called the Feynman gate or Controlled-Not gate - CNOT). Logically, the CNOT gate can be described by two equations: $P = a$, $Q = a \oplus b$, where bit *a* is referred to as the control bit and *b* is the controlled or data bit. Input variables are *a* and *b*, output variables are *P* and *Q*. When the control bit *a* has value |0> the gate does nothing, the controlled bit *b* has the same value on input and on output. However, when the first bit *a* has value |1>, the controlled bit *b* is inverted (since $1 \oplus b = b'$). Therefore this gate is called Controlled-NOT, which means that the controlling bit controls the NOT operation.

This gate can be extended to a ternary Toffoli gate with GF(3) operations replacing GF(2) operations as shown in Figure 2 and used to build ternary butterflies for ternary Reed-Muller expansions and transforms. A generalized Toffoli gate can be built (in binary or MV logic) where the number of control inputs is arbitrary. The Toffoli gate is also called the Controlled-Controlled-NOT, since the NOT operation is controlled by a product of inputs *a* and *b*. Feynman, Fredkin and Toffoli gates, named after their inventors, are examples of controlled gates. Controlled gates are also created for multi-valued and hybrid quantum circuits. In hybrid logic, a ternary bit *A* may control a binary bit *B* or a binary bit may control a

quaternary bit, but every "wire" is for one radix logic only, binary, ternary or quaternary throughout the entire circuit. Each "wire" in such a circuit represents an instance of a qubit (or qudit) in a quantum circuit diagram. We have designed quantum oracles for all these types of gates.

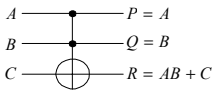


Figure 2: A 3*3 Generalized Toffoli Gate. Symbol + stands for GF(2) and GF(3) addition for binary and ternary quantum case respectively

It is a significant fact that the unitary quantum gates such as CNOT can realize any quantum logic function (including standard binary). Every non-reversible Boolean or multiple-valued circuit C_1 (with arbitrary numbers of inputs and outputs) can be transformed to a reversible circuit C_2 (described by a permutation matrix).

3. Proposed Architecture

In this section, we introduce the proposed architecture for finding minimum FPRM

3.1 Architecture

The proposed architecture is shown in Figure 3. There are 4 blocks in the architecture, *FPRM processor*, *Cost Function and Comparator*, and corresponding inverse blocks *Inverse FPRM processor*, *Inverse Cost Function and Comparator*.

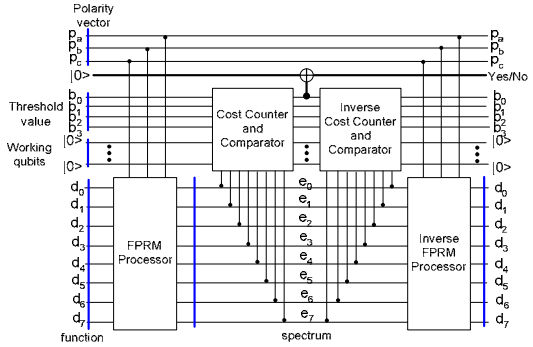


Figure 3: Quantum Architecture for FPRM Oracle for Grover's Method

The input of the *FPRM processor* is the vector of minterms (true or false) for a given Boolean function and the polarity vector. The output of the FPRM processor is the RM spectrum coefficients for the given Boolean function and polarity specified by (p_a, p_b, p_c). The detailed architecture for the *FPRM processor* is presented in the next section.

There are two input busses for the *Cost Function and Comparator* block, these are for the threshold value and the polarity vector. The FPRM processor requires the 2^n truth vector of the Boolean function and produces 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 (P) in the vector of spectral coefficients is counted. Second, a comparison of P with the threshold value (T) is accomplished. If $P \le T$, 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 of these function blocks.

3.2 Grover's Algorithm

In the architecture of Figure 3, we use Grover's algorithm to evaluate the condition $P \le T$. In the following, we give a brief introduction to Grover's algorithm.

Assume you have a system with states $N = 2^n$ labeled $S_1, S_2, ..., S_N$. These 2^n states are represented by n-bit strings. Assume there is a unique marked element S_m that satisfies a condition defined as C such that $C(S_m) = 1$, and for all other states $C(S) = 0$. Assuming that C can be evaluated in unit time. Grover's algorithm can minimize the number of evaluations for C [3]. A quantum register is just a group of qubits, all part of the same quantum mechanical system. Just as an n-bit register is capable of representing 2^n distinct values, so too will an n-bit quantum register assume one of the 2^n basis states when measured [1].

The idea of Grover's algorithm is to place a register in an equal superposition of all states, and then selectively invert the phase of the marked state, followed by inversion about the mean operation a number of times. Selective inversion of the marked state, followed by the inversion about the mean is also referred to as the Grover Operator. 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 one [2]. Observe that we did not modify Grover's algorithm at all in this work, we just showed that it can be extended by a special way of building the oracle. This is true for many other applications as well.

Placing the register in an equal superposition of all states can be accomplished by applying the Walsh-Hadamard operator [3]. 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 [4].

Applying the inversion about the mean is equivalent to applying an operation whose matrix representation is: $A_{ij} = 2/N$ if $i \neq j$ and $A_{ii} = -1 + 2/N$ to the quantum register [3]. A block diagram for a quantum circuit implementation of Grover's algorithm is shown In Figure 4.

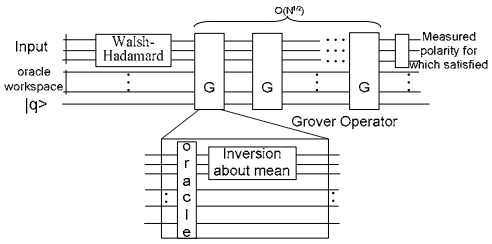


Figure 4: Grover's Algorithm Block Diagram

4. Circuit Design

In this section, we present the detailed implementation for blocks described in the overall architecture.

4.1 FPRM Processor

The butterfly diagrams described previously 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 5 contains the butterfly diagrams for 1-variable functions. The diagram on the left represents the polarity-0 transform while that in the middle represents the polarity-1 transform. Values d_1 and d_2 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 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.

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 5 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.

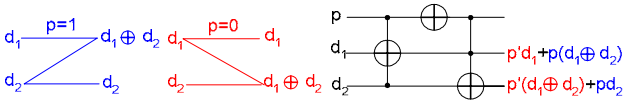


Figure 5: RM Transformation Butterflies and Corresponding Quantum Logic Circuit

Figure 6 shows the 3-variable FPRM processor. There are 3 polarity bits and 8 input lines for a 3-variable processor. There are 3-levels in the processor that correspond to a 3-level "butterfly" diagram such as those shown in Figure 1.

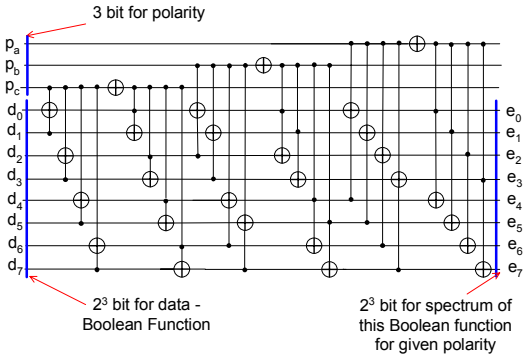
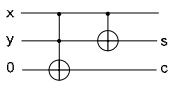


Figure 6: 3-variable FPRM Processor

4.2 Cost Counter and Comparator

There are two tasks to be accomplished in this block. The first task is to count T, while the second task to evaluate the condition $P \leq T$. If $P \leq T$, the circuit will output one, otherwise zero.

The "count ones" function can be accomplished using a tree of half-adders and full-adders and is also known in the arithmetic community as an 8:4 compressor. The quantum implementation for the half-adder is shown in Figure 7 and the full-adder is shown in Figure 8.



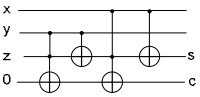


Figure 7: Half-adder **Figure 8: Full-adder**

The 8:4 compressor based on a tree of full-adders and half adders is shown in Figure 9. The compressor circuit has 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 3-bit value. The detailed quantum implementation is shown in Figure 10 (in the leftmost box). This result along with the threshold value then serves as input to the comparator.

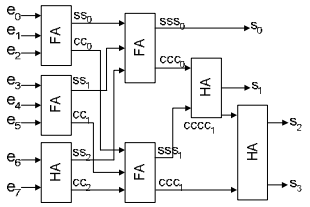


Figure 9: 8:4 Compressor Tree

To build the comparator, we first derive the Boolean function for a comparator that compares two unsigned 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 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, we design the quantum circuit as shown in Figure 10 (in the rightmost box). 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 Toffoli, generalized Toffoli, Feynman, and NOT gates are their own inverses. This is not true in ternary logic, but designing inverse circuits is also straightforward in this logic [5].

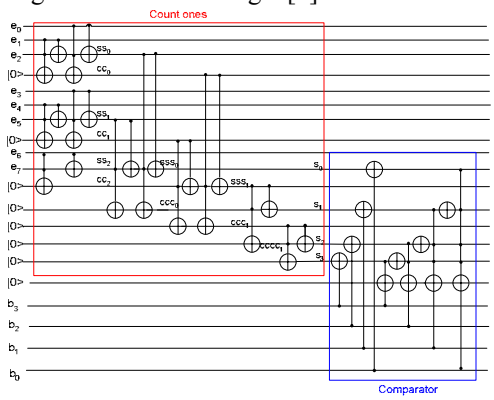


Figure 10: Quantum Implementation of 8:4 Compressor and Comparator

5. MVL Compressor Tree Implementation

As described in the previous section, this 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.

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 (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 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 wordlength and can be used as the basis for other high-speed arithmetic circuits. Table 1 first appeared in [23] and illustrates how the redundant digit set is exploited to prevent long carry ripples.

Table 1: Signed Binary Addition Table

$a_i + b_i$	Sign info. of digits in pos. *i*-1	c_{i+1}	s_{i+1}
$\overline{1}+\overline{1}$	Not Used	$\overline{1}$	0
$\overline{1}+0$	Either is Neg.	$\overline{1}$	1
$\overline{1}+0$	Neither is Neg.	0	$\overline{1}$
$0+0$	Not Used	0	0
$1+\overline{1}$	Not Used	0	0
$1+0$	Either is Neg.	0	1
$1+0$	Neither is Neg.	1	$\overline{1}$
$1+1$	Not Used	1	0

In Table 1, 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 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 generalized Toffoli gates as shown in upper-left box in Figure 11.

Based on Table 1, the sum and carry values are:

$$A^{\{0\}} + B^{\{0\}} + A^{\{0\}} B^{\{0\}} = s_2^{\{2\}}$$
$$2 \bullet s_2 = s_1$$
$$C = S(2 \bullet (A_i \oplus B_i)) = S \oplus (2 \bullet (A_i \oplus B_i))$$

where $\oplus, \bullet$ are GF add and multiplication respectively.

From the formulas, the entire adder can be implemented as shown in Figure 11. 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.

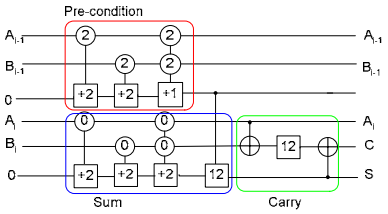


Figure 11: Sign Adder Quantum Implementation

6. Applications

When we write "applications" we mean of course only hypothetical applications because quantum computers are not available yet. In this sense, applications can be all those in which the FPRM transform is used, including logic design (also logic minimization for reversible and quantum circuits themselves), image processing and DSP, and machine learning. There is also a quickly developing area of quantum game theory where many known results for quantum algorithms find very interesting interpretations. For instance, the problem discussed above is more general than the game of finding the conjunctive formula of literals for a given set of data. Our machine 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. All circuits presented here can be generalized to ternary quantum gates [5,15], allowing for ternary butterflies [22] and more efficient arithmetic for larger counters and comparators.

7. Conclusions and Future Work

We showed how Grover's algorithm can be used to solve a practical problem in classical logic minimization. This work also illustrates the design of practical reversible circuits using quantum gates for blocks that will be normally used inside oracles. In many oracles that we tried, there always exist arithmetic blocks such as adders, subtractors, comparators, counters of ones (compressors), and logic blocks. Here we add one more type of block, butterfly transforms. Our goal is to simulate this algorithm using QuidPro [14] to practically validate its correctness. The next goal is to design and simulate similar algorithms for Galois Fields other than GF(2) [15], for ESOP and for Karhunen-Loeve transforms [10]. We also intend to build quantum oracles for graph coloring, satisfiability, Hamiltonian path and other intractable problems for both binary, ternary and hybrid binary/ternary algorithms/circuits. One of the aims of our work is to discover practical and efficient methods of designing and laying out binary, multi-valued and hybrid quantum circuits not for random benchmarks but for practical blocks and architectures.

8. References

[1] C. Williams, S. Clearwater, Explorations in Quantum Computing, *Springer-Verlag*, New York, Inc.

[2] 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

[3] M. Hayward, Quantum Computing and Grover's Algorithm, http://alumni.imsa.edu/~matth/quant/473/473proj/node1.html

[4] E. Abe , School on Quantum Computing @Yagami, http://www.appi.keio.ac.jp/Itoh_group/members/abe/qis-4.pdf

[5] M. H. A. Khan and M. A. Perkowski, "Quantum Realization of Ternary Parallel Adder/Subtractor with Look-Ahead Carry," *Proceedings of the International Symposium on Representations and Methodology of Future Computing Technologies*, 2005.

[6] N. Song, M. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Functions," *IEEE Transactions on Computer Aided Design*, Vol. 15, No. 4, April 1996, pp. 385-395.

[7] T. Sasao, M. Fujita, Representations of Discrete Functions, *Kluwer Academic Publishers*, 1996

[8] 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.

[9] M. Thornton, R. Drechsler and D. M. Miller, Spectral Techniques in VLSI CAD, *Kluwer Academic Publishers*, 2001

[10] M. Thornton "The Karhunen-Loève Transform of Discrete MVL Functions," *IEEE International Symposium on Multiple-Valued Logic* (ISMVL), May 18-21, 2005.

[11] B. Patterson, http:www.cs.iastate.edu/~patterbi/cs /quantum.fp/FinalPaper.pdf

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

[13] G.F. Viamontes, M. Rajagopalan, I.L. Markov and J.P. Hayes, Gate-Level Simulation of Quantum Circuits, *quant-ph/0208003*.

[14] 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

[15] 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.

[16] U. Kalay, M. Perkowski, D. Hall, "A Minimal Universal Test Set for Self-Test of EXOR-Sum-Of-Products Circuits," *IEEE Tr. on Computers,* March 2000, Vol. 49, pp. 267 - 276.

[17] S. Reed, "A class of multiple error correcting codes and their decoding scheme," *IRE Trans. Inf. Th.*,Vol. PGIT-4, 1954, 38-39.

[18] I. Zhegalkin, "O tekhnyke vychyslenyi predlozhenyi v symbolytscheskoi logykye," *Math. Sb.*, Vol. 34,1927, 9-28, (in Russian).

[19] 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*, vol. 12, No. 5, May 1993, pp. 621-632.

[20] A. Mishchenko and M. Perkowski, "Fast heuristic minimization of exclusive-sums-of-products", *Proc. Reed-Muller Workshop '01*, pp. 242-250.

[21] J. W. Cooley, and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series. *Math. Computation*, Vol. 19:297-301, 1965.

[22] Ch. 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.

[23] Y. Harata, et.al ,"A High-Speed Multiplier Using a Redundant Binary Adder Tree,"IEEE J. Solid-State Circuits, vol. 22, pp. 28-34, Feb. 1987.