Synthesis of reversible circuits with small ancilla bits for large irreversible incompletely specified multi-output Boolean functions

Christopher J. Stedman, Bruce Yen and Marek Perkowski

Department of Electrical and Computer Engineering, Portland State University,
P.O. Box 751, Portland, Oregon, 97207-0751,
cstedman@cs.pdx.edu, byen@pdx.edu, mperkows@ece.pdx.edu

Abstract

The paper presents several improvements to the well-known MMD algorithm of Miller, Maslov and Dueck for binary reversible cascade synthesis. First, we present an improved version of the first phase of MMD by extending the single ordering of input minterms during the solution search into multiple orderings. The results are better than in the original MMD without template matching. Next a novel logic synthesis method for binary reversible circuits with small ancilla bits is presented that can be applied to large functions that are originally irreversible; conversion to reversible is thus a part of the method. Internally it uses two programs: MMD and the ESOP minimizer Exorcism. The synthesis method is algorithmic and simple; optimization is reduced to search with cost function minimization. The method is applied to incompletely specified multi-output Boolean functions.

1. Introduction

Reversible logic and quantum computing are recently flourishing research areas. Reversible circuit design and optimization are a part of quantum circuits design. Reversible circuits have been also shown to potentially reduce power consumption, regardless of the technology [1]; this will become more practical in new realizations of reversible logic, such as quantum. The MMD algorithm *(Miller, Maslov and Dueck)* is currently the leading reversible logic synthesizer [2]. This program assumes a reversible function specification as data and it uses no ancilla bits. MMD software is reasonably fast and it distinguishes itself among other programs of this type because it achieves (theoretical) 100% convergence regardless the problem size [2]. Practically it can be applied to 8*8 variable reversible functions and some special larger functions. This program is therefore the current benchmark for the evaluation of programs for reversible circuit synthesis. Due to MMD non-minimal results, several research groups are constantly attempting to improve its algorithm. Agrawal and Jha's algorithm uses the number of terms in the Positive Polarity Reed-Muller (PPRM) expansion of synthesized functions as its cost function [3], and Kerntopf's algorithm uses complexity of SBDD's as its cost function [4]. The use of complexities of ESOPs, FPRMs and Maitra cascades in the cost functions that evaluate the search results have been also proposed in the newer versions of composition-based search approaches [9, 10, 11, 12].

The weaknesses of MMD include: (1) for n-variable functions it uses a permutation vector of length 2^n as its input data which precludes from using it for large circuits, (2) It works only with completely specified functions, thus excluding initial specifications being relations or incompletely specified functions, (3) It does not allow to create arbitrary orders of output functions, which would be one more degree of freedom and is useful in some problems of quantum layout-level optimization [7], (4) It needs template matching method to optimize its results because only one order of realizing minterms is used in it and the initial result may be far from minimum, (5) It does not allow to investigate the trade-off between the number of ancilla bits and the cost (length of quantum cost or a gate cost) of the cascade. Below we present our preliminary research on improving the MMD's weaknesses (see also [7]).

The structure of this paper is the following. Section 2 presents a new ordering algorithm to replace the natural binary ordering of MMD. Section 3 introduces new search strategy that uses the so-called EXOR Lattices to decompose the output functions and convert their

irreversible to reversible forms. Section 4 concludes the paper.

2. MMDS Ordering

In this section, MMD's natural binary order is challenged as the only 100% convergent order. It is found that MMD's order falls into a subset of orders that do not exhibit certain important property that we call "control line blocking". This observation leads to the creation of what we call the "MMDS ordering". Without any backtracking, bi-directional search or template matching the new ordering MMD algorithm uses multiple MMDS input orders to produce better results than the original MMD ordering. It can be used with any number of inputs and has larger gains compared to MMD when the number of inputs increases.

2.1. Background

The MMD algorithm transforms step-by-step a reversible function to its identity function. The function is arranged in a natural binary code order by inputs assignments. Each iteration adds a gate in order to correctly transform the outputs to equal the inputs without changing any of the previously assigned output patterns (minterms). Gates are chosen to reduce the cost function such as a Hamming distance of the gate choice function to the original function or to identity function. In some variants the gates can be added bi-directionally, at the beginning and the end of the cascade. Once a complete circuit is generated, the original template matching approach is applied to reduce the gate cost [2], which is a variant of local optimization method. The advantages of the MMD include: (1) no garbage and thus no ancilla bits, (2) an advantage of using high-quality template matching to optimize the initial result, (3) 100% convergence. The shortcomings of the original MMD include: (1) poor heuristic to select gates, (2) using only one input order, (3) using gates that are not realized in practice and are macros rather than quantum realizable gates, thus may be very expensive for circuits with many inputs (like an 8-input Toffoli gate).

MMD's positive characteristics, including no garbage and 100% convergence, are the reason for it being the leading reversible logic synthesizer. MMD's shortcomings make it easy to appreciate modifications where a small improvement in algorithm could make a huge improvement in the reversible circuit's quality.

2.2. Algorithm

The goal of this section is to improve the MMD algorithm. Many attempts have been made at improving the gate-selection heuristics. These attempts resulted in very good/minimal solutions for some circuits or non-converging/incorrect circuits for others [4-12]. Thus the order of selecting outputs to be covered by gates was decided to be even more important than the gate –choice heuristics. Since the gate choice heuristic is currently being pursued elsewhere, the goal of this section is to explore whether or not other input orders can be used with 100% convergence. Figure 1 is an example of two input orders that represent the same function. If the algorithm doesn't converge 100% of the time it is inferior to MMD. Is one order better than the other? MMD's premise is built on the idea that the natural binary order is the best order. The fact that this order converges for all output permutations creates a very convincing argument that this order is one of a kind.

Input variables: a, b, c		
Output variables: a, b, c		
000	->	111
001	->	110
010	->	011
011	->	000
100	->	100
101	->	010
110	->	101
111	->	001
Input variables: a, b, c		
Output variables: a, b, c		
000	->	111
100	->	100
010	->	011
001	->	110
110	->	101
101	->	010
011	->	000
111	->	001

Figure 1. Example of multiple input orders

All examples here will use three-input functions, but all theories apply to reversible functions of arbitrary size. Since we wanted to investigate the reason behind using the MMD order, the exact opposite order was tested [111, 110, 101, 100, 011, 010, 001, 000]. For 100% convergence, any output permutation can be applied to this input order. The first output can easily be changed to match its input, but to change any following outputs would result in changing the first output. This order conflicts with the main MMD's idea [2] of not changing any previously set outputs. This idea is also what guarantees MMD's convergence, so the above order is not acceptable. Using this order as a guide to what is unacceptable, it can easily be seen that 111 can not precede any other number because it would block all of the control lines of the following numbers.

Definition 1. Control Line Blocking condition occurs when all control lines are a subset of the control lines of an output previously set in the input order.

When this condition occurs it makes it impossible to change any output bits during the current iteration without altering the output bits which have been set previously. Occurrence of this condition leads to the inability to converge.

Mathematical Check =>
if #later = #later & #earlier
then there is control line blocking
Example of control line blocking
101 = 101 & 111
Example of no control line blocking
001 = 101 & 011

Therefore, any order of inputs that does not lead to the occurrence of blocking condition can be used in an improved MMD algorithm. The software to find all non-blocking permutations for any number of inputs can be found in [5]. No control line blocking seems to be a very restrictive rule. For a three-input function there are initially 8! (40,320) permutations. And thus as many orders. Instantly that number is reduced to 6! Since 000 must come first and 111 must come last. Using the software found in [5], 48 permutations, called MMDS orders, were found to exhibit no control line blocking for all 3*3 reversible functions. Included in this set is the original MMD ordering. As the number of input lines increases, the number of non-blocking orders increases exponentially. Four inputs have 78,880 different non-blocking permutations. As the amount of non-blocking orders increase so does the optimality of the MMDS ordering.

Crucial in implementing MMDS ordering was the modification to the naive control possibility function written for MMD *{set_controls()}*. (MMDS ordering code was developed by modifying the MMD code [5]). The added degree of freedom of multiple input orders could not be handled by the old control possibilities function, because it assumed the input order to be the MMD order. A set of rules were created to choose the possible control choices from the set of all possible control line choices.

1. The target bit can not have control,
2. Keep only controls that flip the target bit,
3. No past outputs can be changed.

The new *set_controls()* function can be found in [5].

Examples of Control Possibilities Choice Rules:
Ex. 1. All Controls [000, 001, 010, 011, 100, 101, 110, 111]. Solution is shown in Figure 2.

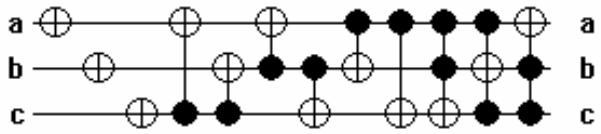

Figure 2. Quantum array for Example 1.

Ex. 2. Target bit can not have control => Target = 001 → {0,2,4,6}. Solution is shown in Figure 3.

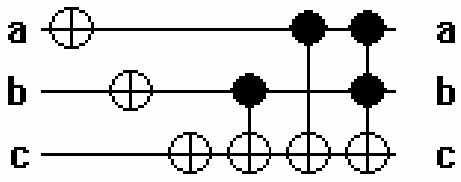

Figure 3. Quantum array for Example 2.

Ex. 3. Keep only controls that flip the target bit. Solution is shown in Figure 4.

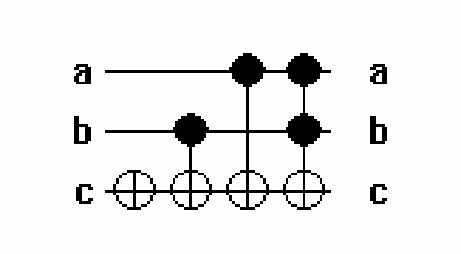

Figure 4. Quantum array for Example 3.

Ex. 4. No past outputs can change (varies with all examples).

The control possibilities are then sent to the gate choice function to produce a circuit. Currently gate choice is based on Hamming distance but it can be any cost function [4, 9,10,11,12].

Using control line blocking as the only rule, a subset of all input orders can easily be found, and it can be easily proven that all non-blocking input orders will converge for all output permutations.

Theorem 1.
All non-blocking input orders converge for all output permutations.

Proof of Convergence:
Convergence is guaranteed in MMD and MMDS because at any given point in the algorithms all following output bits are able to be changed without altering any previously set outputs. This is guaranteed because the input orders do not exhibit control line blocking. With MMD and MMDS' methodical approaches, as long as all output bits can be changed without altering any previously set outputs these algorithms will converge every time.

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

2.3. Results

Claim 1
No input vector order is better than the rest for all permutations:

This claim is based on a sample of input orders tested with all possible output permutations (i.e. all 3*3 reversible functions as in [2]. As can be seen in the results below, the average gate size is approximately the same for each input order (WA = 7.3), so no apparent advantage can be seen by using one order over another.

Test 1. MMD Order {0,1,2,3,4,5,6,7}
COST statistics: sum = 294971, total = 40320,
WA = 7.31575
SIZE statistics: sum = 294971, total = 40320,
WA = 7.31575

Test 2. MMDS Order {Random1}
COST statistics: sum = 290593, total = 40320,
WA = 7.20717
SIZE statistics: sum = 290593, total = 40320,
WA = 7.20717

Test 3. MMDS Order {Random2}
COST statistics: sum = 298145, total = 40320,
WA = 7.39447
SIZE statistics: sum = 298145, total = 40320,
WA = 7.39447

For all orders the results are.

#of gates	SIZE	COST
j = 0:	1	1
j = 1:	12	12
j = 2:	90	90
j = 3:	457	457
j = 4:	1626	1626
j = 5:	4140	4140
j = 6:	7696	7696
j = 7:	10143	10143
j = 8:	9062	9062
j = 9:	5164	5164
j = 10:	1684	1684
j = 11:	218	218
j = 12:	27	27
j = 13:	0	0
j = 14:	0	0
j = 15:	0	0
j = 16:	0	0
j = 17:	0	0

COST statistics:
sum = 284478 total = 40320 WA = 7.05551

SIZE statistics:
sum = 284478 total = 40320 WA = 7.05551

Every dot below represents that 500 operations have been done. The number after it is the total difference between the number of gates. MMDS reduces the number of gates by 38492 for all 40320 functions. This is almost one gate per function but the improvement is only approx .3 in WA.

```
.278.601.1243.1620.2196.2663.3081.
3669.4330.4945.5246.5574.6042.6481
.7096.7506.056.8638.9293.9881

.10446.10904.11353.11641.12113.1
2790.13508.14055.14464.14908.153
58.15757.16316.
6554.16997.17651.18277.18738.191
01.19595

.20176.20722.21077.21760.22336.2
2991.23369.23745.24277.24750.251
00.25577.26092.
6665.27274.27938.28294.28577.289
19.29625

.30110.30628.31034.31546.32052.3
2491.33155.33531.33938.34259.346
13.35087.35495.5870.36261.36721.
37196.37564.38104.38412
38492
```

Claim 2.

Using all possible input orders produce more optimal answers without template matching and bi-directional gate addition:

Since MMD order is a subset of MMDS, MMD results will be improved. As the number of inputs increases, MMDS ordering produces substantially better circuits.
Some examples of Results:

Input variables: a, b, c		
Output variables: a, b, c		
000	->	000
001	->	100
010	->	101
011	->	001
100	->	110
101	->	010
110	->	011
111	->	111

Figure 6: Random Reversible Function

Figure 5 shows a randomly generated reversible function. Figures 7 – 11 illustrate various variants of its design using MMDS.

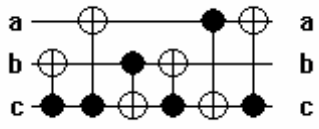

Figure 6: Circuit created with MMD using No Bidirectional & No Templates

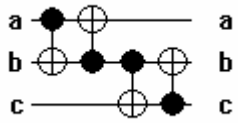

Figure 7: Circuit created with MMD using Yes Bidirectional & No Templates

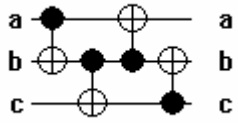

Figure 8: Circuit created with MMDS using No Bidirectional & No Templates Input Vector{40} = {0,4,2,1,3,5,6,7}

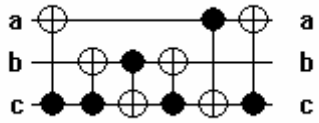

Figure 9: Circuit created with MMD using No Bidirectional & Yes Templates

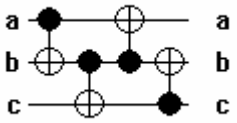

Figure 10: Circuit created with MMD using Yes Bidirectional & Yes Templates

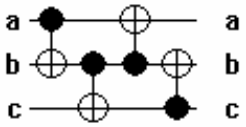

Figure 11: Circuit created with DMMD using No Bidirectional & Yes Templates. Input Vector{32} = {0,2,4,6,1,3,5,7}

3. Synthesis of Binary Reversible Circuits With Small Ancilla Bits Using EXOR Lattice for Incompletely Specified Functions

As mentioned in section 1, the MMD method is unable to generate circuits for non-reversible functions. By adding as many ancilla bits as outputs of the initial irreversible specification it is possible to extend the MMD method to non-reversible functions by converting the initial input/output specification to a reversible function specified as a permutation vector. However, as MMD method can only be applied to completely specified functions, the task of optimally specifying the added ancilla bits is non-trivial.

In the method presented here, garbage bits are considered to be non-restrictive, and are introduced freely to reduce the complexity of synthesis. Future quantum computers (such as those used for integer factorization) will be useful only for a large number of lines (qubits). Some techniques such as mirroring and constant overlap [7] will allow to locate reversible permutative oracles inside a larger two-dimensional space similarly as it is done in current VLSI: the specification is partitioned for logic minimization and the synthesized circuits are placed and routed. This general methodology justifies our more generous use of ancilla bits than in MMD and previous synthesis algorithms.

3.1. Synthesis based on Exclusive-OR Lattices

The nature of an EXOR logic gate gives the following useful property: given a set of three logic functions A, B, and C, such that A XOR B = C, it is obvious that B XOR C = A and A XOR C = B. Therefore, given any two functions, the third of this closed set of functions can be uniquely determined. The presented below algorithm exploits this property by performing a logical EXOR between all output functions to search for functions that are commonly repeated or easily implemented. Furthermore, the property scales to any number of functions, producing the new concept of an EXOR lattice, as shown in Figure 12.

In the method presented here, garbage bits are considered to be non-restrictive, and are introduced freely to reduce the complexity of synthesis. Future quantum computers (such as those used for integer factorization) will be useful only for a large number of lines (qubits) [7].

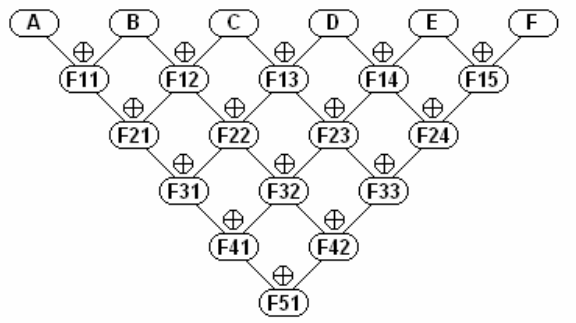

Figure 12. Two-Dimensional EXOR Tree

The top row of nodes in Figure 12 represents outputs of 6 different functions. Each subsequent row represents the logic functions obtained by logically EXOR'ing the previous row's functions with one another. Therefore, by selecting functions A, F11, and F21, we can implement B directly with a logical EXOR of F11 with A. Performing an EXOR of F11 and F21 we obtain F12, which can be used to obtain output C, as well. In this case, this map is 2-dimensional (planar). However, when creating such a DAG, the first and third node may have some logical EXOR that can be used instead to obtain the third node, rather than having to implement the second to implement the third. This would result in a multi-dimensional DAG (which is a lattice in which the nodes are EXORs of functions being all subsets of the set {A,B,C,D,E,F}).

An example of the implementation is given in Figures 13 – 15. In this example, there is an initial phase to create the setup functions, which will be treated as the functions found in the top row of Figure 12. Figure 13(a) is the set of Karnaugh maps of desired output functions A, B, and C. The setup functions A', B', and C' need to be created as a result of the method of synthesis via quantum cascades (note that A' does not denote a negation of variable A). These setup functions are created by EXORing the desired output function with the function initially on the input line. For example, because function A will be implemented on the line that has initially value *a*, we perform a logical EXOR of the desired function A, with the initial function A to generate setup function A', as shown in Figure 13(b). Figure 14 shows the logical functions in their EXOR Lattice form.

By implementing functions F1 and A', B' can be realized by F1 EXOR A'. Furthermore, F3 EXOR F1 gives a function F2, such that F2 EXOR B' realizes C'. Therefore, by implementing A', F1, and F3, functions A', B' and C' can be realized through a series of EXORs. Notice that F1 and F3 are both

rather easily implemented, whereas implementation of B' and C' would have been more costly. Coverings are first found for A', F1, and F3 by using an EXOR synthesis tool such as EXORcism. Such coverings are shown in Figure 14.

Let us denote by ~x the negation of variable x. The A' is given by "(a) XOR (~a)(b)(c)," implementation is straightforward, and given in Figure 15(a). Figures 15(b, c*)* implement F1 and F3, respectively. These can be trivially cascaded, as shown in Figure 15(d). To implement the remainder of the lattice, the lowest row is implemented first, as in Figure 15(e), where F2 is implemented from an EXOR of F1 and F3. As functions A', F1, and F2 now exist, the next row to be implemented is B', and C', which are implemented in Figures 15(f) and 15(g). Finally, Figure 15(h) realizes the final expected functions A, B, and C, from setup functions A', B', and C'. The complete cascade is shown as Figure 15(i).

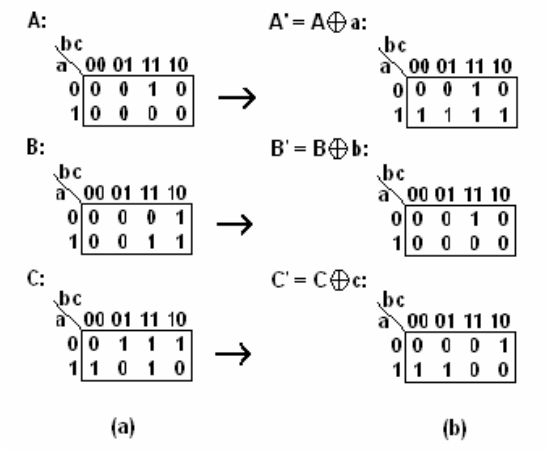

Figure 13. Setup Functions

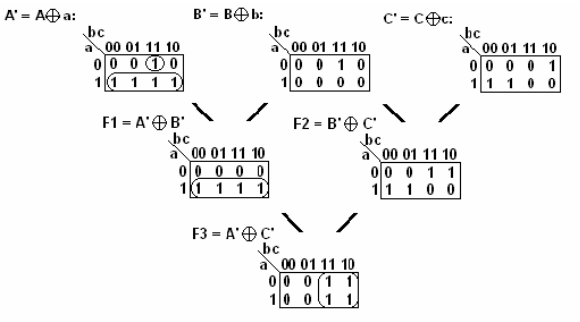

Figure 14. EXOR Lattice

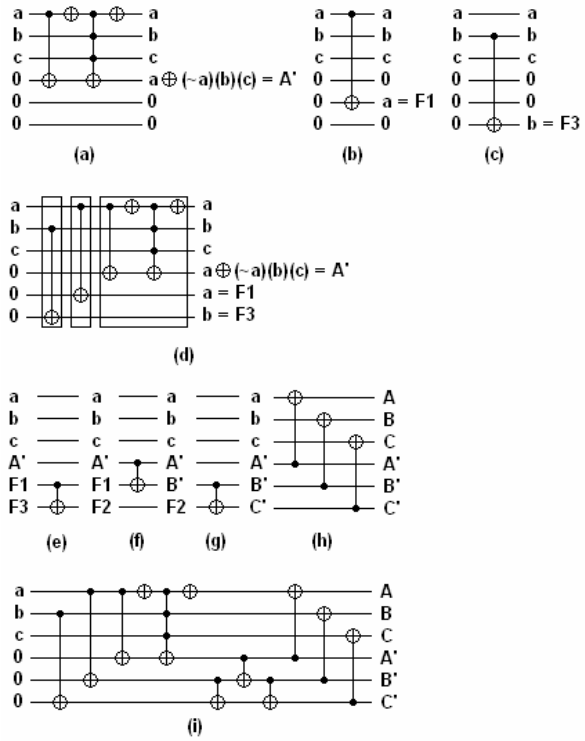

Figure 15. Implementation of Exor Lattice.

The EXOR lattice can be implemented as several smaller lattices, rather than one large lattice as has been shown in Figure 12. Note that the maximum necessary garbage inputs correspond directly to the number of the necessarily implemented lattice nodes. Under worst case, this will result in N additional garbage inputs for an N input, N output vector. In Figure 16(a), a lattice is shown where black and grey nodes represent functions that are either easily implementable or are repeated functions. Each shaded node selected as an initialization node (i.e., A', F1, F3 nodes, as from the previous example) either results in a reduction of complexity in implementation of the said node, or even allows ignoring the node altogether, if the node is either easily implemented or repeated, respectively.

In the example of Figure 16(a), two smaller trees are selected over the larger tree in Figure 16(b). The solution can be further reduced, as in Figure 16(c) by recognizing that overlapping lattices result in no added overhead. This reduces the solution found in Figure 16(b) by one node, replacing the node by a shaded node. Choosing the smaller trees give the benefit of less nodes to traverse in order to implement the highest row nodes. The special case of no easily implemented function nodes or no

repeated function nodes would result in a special case where the initial nodes may be best selected as the top row. In this case, the setup functions are directly implemented; this case is equivalent to a direct implementation of logic functions onto constant input lines.

Clearly, choosing the best EXOR lattices to implement is a non-trivial task, but can be simplified with good representation that can identify functions that are similar or dissimilar to one another. By grouping similar terms together, good heuristics can be used to relate similar node functions in an EXOR lattice.

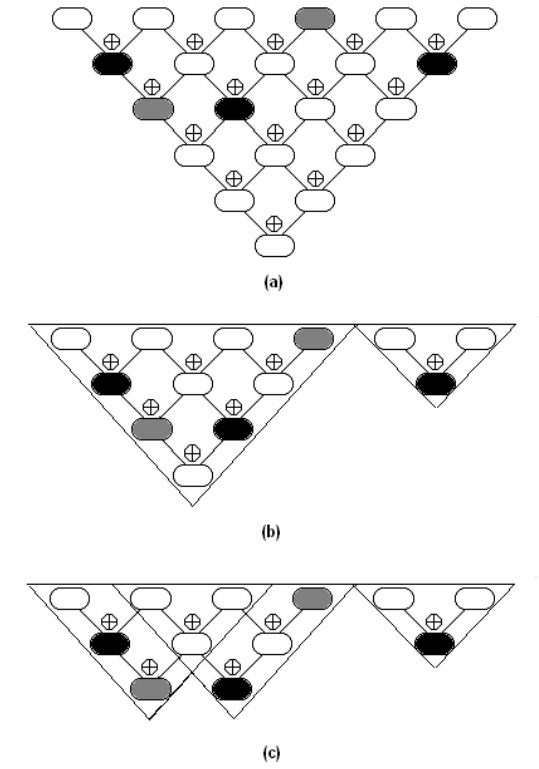

Figure 16. Exor Lattice Selection

In the MMD algorithm, incompletely specified functions must be completely specified due to the necessity of ordering in the output vector. Optimization of the complete specification is a non-trivial and difficult task. However, in the EXOR lattice method, it is possible to trivially synthesize quantum cascades for incompletely specified functions without specifically defining don't-cares. An example is given in Figure 17. Furthermore, by introducing the N constant-input lines, it is also trivial to implement irreversible functions. For example, Figure 17, if we remove the variable "c" altogether (thereby removing the don't-care states), output function "A" is irreversible. Adding no additional overhead, the output vector can be implemented in exactly the same fashion. Final quantum array is shown in Figure 18.

As the method has no dependency on ordering, the output vector needs not be of a fixed order; it can be any permutation of given outputs. This freedom introduces yet another degree of optimization; certain inputs selected for particular output lines may result in further optimization possibilities by changing the setup function set. This problem, however, is, as of yet, beyond the scope of this paper. It should be noted, in the example of Figure 17, that the implementation of each setup function is not explicit; as each EXOR minterm of the setup function is realized, it is directly EXOR'ed to the expected output line. This is demonstrated as a different 'flavor' of synthesis, which may be necessary in some cases, when some function needs to be copied several times.

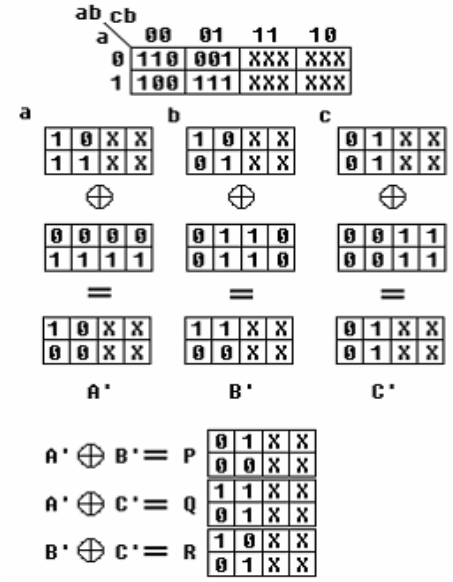

Figure 17. Incompletely Specified Function.

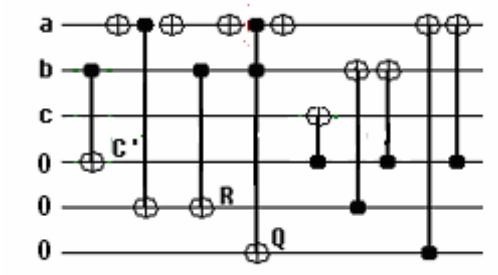

Figure 18. Quantum Cascade for the incompletely specified function from Figure 17.

Lastly, it should be noted that, though we previous ly mentioned that this method assumed N input, M output, 1≤M≤N, it is possible to use the ancilla bits to synthesize as many outputs M such that 1≤M≤2N. One special case worth mentioning is when the first N output functions (N≤M≤2N) of M output functions are a reversible vector of outputs, and implemented on the non-constant input lines, and the remaining M-N outputs on the constant input lines. In this special case, because the first N outputs are a reversible set, it will be possible to directly implement the remaining M-N output set on the constant input lines. And, because this method allows for freedom of output permutation, it is a viable method of synthesis to select outputs in such a way. The new version of MMD is called by this procedure.

3.2. Reversible Cascade Synthesis for Irreversible Incompletely Specified Functions.

(1) Read incomplete functions.
(2) Create the Setup Functions (*Figure 13*)
    a. Output lines are selected for each output function. Setup functions are calculated for each output.
    b. Coverings are developed for each setup function, using the EXOR synthesis tool, EXORcism [16].
(3) Create EXOR Lattice
    a. Ordering of variables by the Positive Polarity Reed Mueller. This is used to optimize the search for similar functions.
    b. Development of an EXOR Lattice
    c. Cost Functions
        i. Easily implemented functions (Variable)
        ii. Repeated Functions (0, after the first implementation)
        iii. Every node necessarily traversed (Cost of 1 (initially not implemented) for each EXOR to generate this function)
        iv. Initial implemented functions (Variable)
(4) Initial Nodes *(Figure 15(a, b, c, d), Figure 16)*
    a. Initial nodes are selected based on:
        i. Repeated Functions
        ii. Easily Implemented Functions
        iii. Cost Functions
    b. Direct synthesis of initial nodes to the quantum cascade.
(5) Traversal of the EXOR Lattice (*Figure 15(e, f, g)*)
    a. Search and subsequently synthesize lowest-row, not-yet-implemented node with two adjacent, implemented nodes.
    b. Iterate, to traverse the entire EXOR lattice until top row is completely synthesized.
(6) Use setup Function to Output Function Synthesis. (*Figure 15(h)*)
(7) Call MMD with MMDS ordering to improve parts (in windows) of the array or the entire quantum array.

4. Conclusions

In this paper we have presented a new approach to synthesize binary reversible cascades for incompletely specified functions. Interfacing 'real world'-interpretable logic with probabilistic spirit of quantum computing requires some form of intermediate description language, for instance multiple-valued logic. In this paper, binary logic is used as the most approachable, both by reason of the reduction of complexity in only 2-states, and due to the maturity of binary logic in standard logic synthesis. However, all the presented methods are general and can be relatively easily adapted to quaternary logic [13]. The reversibility aspect of binary quantum circuits is a necessary characteristic for interfacing between quantum logic and real-world interpretation, as quantum operators (which are what each quantum binary gate is composed of) are intrinsically reversible.

The first goal of this project was to improve on the MMD algorithm. The individual area of improvement was focused on input order. It was found that the natural binary order is only special because it falls in the category of input orders that do not exhibit control line blocking. All orders of this type can be used with 100% convergence in MMDS. This added degree of freedom allows the routine to run through an assortment of input orders until it finds the best circuit for that specific function. Again, different input orders are better suited for distinct functions, so cycling through the all input orders will find the most optimal solution. As the number of inputs increases, the number of input orders increase exponentially. In order to fully utilize this property of multiple input orders, an improved technique must be further developed to find a subset

of all MMDS input orders. For example: by sampling across all outputs then converging in the best area of input orders or by sampling output orders to focus the search in a specific area.

In the second part we discuss a new approach to synthesis reversible functions. Under worst case it introduces N ancilla bits. However, due to the introduction of ancilla bits, this approach is able to synthesize both incompletely specified and irreversible functions without explicit specification or adding any additional ancilla bits to make the function reversible. Given also the freedom of output permutation, we assert that the additional N ancilla bits introduced by this approach more than make up for themselves and present this as an alternative approach to ancilla bit limiting methods for binary reversible logic synthesis.

5. Acknowledgements

The original MMD code [2] was obtained by Prof. Marek Perkowski and his research team at Portland State University from Professor Mike Miller and Dr. Dmitri Maslov. We are also grateful for their comments to our early work.

6. References

1. C. Bennett. Logical reversibility of computation. I.B.M. J. Res. Dev., 17:525-532, 1973.
2. D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In Proceedings of Design Automation Conference, June 2003.
3. A. Agrawal, and N. K. Jha, Synthesis of reversible logic. In Proceedings Design and Test in Europe Conference, Paris, France, February 2004, 1530-1591
4. P. Kerntopf, A New Heuristic Algorithm for Reversible Logic Synthesis. In Proceedings of Design Automation Conference, June 2004
5. C. Stedman, report, PSU, 2005.
6. A. Akashdeep, report, PSU, 2005.
7. V. S. Shivgand, A. Aulakh, and M. Perkowski, Quantum Circuit Layout, submitted to RM 2005.
8. B. Yen, report, PSU, 2005.
9. A. Khlopotine, M. Perkowski and P. Kerntopf, `Reversible Logic Synthesis by Iterative Compositions, *Proc. IWLS 2002,* June 4-7, 2002.
10. A. Mishchenko, M. Perkowski**,** Logic Synthesis for Reversible Wave Cascades', *Proc. IWLS 2002,* June 4-7, 2002.
11. M. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, M. Md. Mozammel Azad Khan, A. Coppola, S. Yanushkevich, V. Shmerko, and L. Jozwiak, ``A General Decomposition for Reversible Logic". *Proc. RM'2001,* August 2001.
12. P. Vetcha, report, PSU, 2004.
13. W. Hung, X. Song, M. Perkowski, Reachability Analysis for reversible minimization. *Proceedings of DAC 2004.* June 2004.
14. M. Miller and D. Maslov, Finding a Set of Optimal 3-line NCV Circuits, LDL 2005.
15. D. Maslov and M. Miller, Reversible and Quantum Circuit Simplification: Circuit -> Circuit, and D. Maslov and M. Miller, Methods for Reversible Logic Synthesis: Creating A Circuit, LDL 2005.
16. A. Mishchenko and M. Perkowski, Fast Heuristic Minimization of Exclusive Sums-of-Products," *Proc. RM'2001 Workshop,* August 2001