Synthesis of Reversible Logic

Abhinav Agrawal and Niraj K. Jha
Department of Electrical Engineering
Princeton University, Princeton, NJ 08544
{aagrawal, jha}@ee.princeton.edu

***Abstract*— A function is reversible if each input vector produces a unique output vector. Reversible functions find applications in low power design, quantum computing, and nanotechnology. Logic synthesis for reversible circuits differs substantially from traditional logic synthesis. In this paper, we present the first practical synthesis algorithm and tool for reversible functions with a large number of inputs. It uses positive-polarity Reed-Muller decomposition at each stage to synthesize the function as a network of Toffoli gates. The heuristic uses a priority queue based search tree and explores candidate factors at each stage in order of attractiveness. The algorithm produces near-optimal results for the examples discussed in the literature. The key contribution of the work is that the heuristic finds very good solutions for reversible functions with a large number of inputs.**

I. INTRODUCTION

Reversible functions have a unique mapping between input vectors and output vectors and are applicable to quantum computing [1], nanotechnology [2], and low power design [3]. Unfortunately, their synthesis eludes traditional methods of logic design. The optimal exhaustive algorithm [4] is too slow. Several different heuristics have been presented in [5]–[7]. Unfortunately, these heuristics do not scale well and require extensive use of template matching. Our algorithm uses an XOR sum of products expression of the output function to synthesize the circuit. Use of such a Reed-Muller expansion of the function was also suggested in [8]. However, their method fails to take advantage of shared functionality among multi-output functions. Our algorithm has several key characteristics. Firstly, it minimizes the number of gates as the primary objective and the size of the gates as the secondary objective. Secondly, it is near-optimal for all 40,320 reversible functions of three variables, and is applicable to functions with a large number of inputs. Thirdly, it does not use the variables and their complements at each stage of the circuit, thus reducing circuit size. Lastly, it does not require output permutation or extra garbage lines (such lines are required to equalize the number of inputs and outputs).

II. BACKGROUND

A function is reversible if it maps each input vector to a unique output vector. A reversible function of n variables can be defined either as a truth table or as a mapping of integers {0,1,...,$2^n - 1$} onto itself. An *irreversible* function can be converted into a reversible function easily. If the maximum number of identical output vectors is p, then $\lceil log_2 p \rceil$ garbage outputs (and some inputs, if necessary) must be added to make the input-output vector mapping unique. There are two main types of reversible gates, the Toffoli gate and the Fredkin gate. An $n \times n$ Toffoli gate, denoted by $TOFn(x_1, x_2, ..., x_n)$, passes the first $n-1$ inputs (referred to as *control* bits) to the output unchanged and inverts the n^{th} input (referred to as the *target* bit) if the first $n-1$ inputs are all 1. A 1×1 Toffoli gate simply inverts the input unconditionally. Using x for input and y for output:

$$y_i = x_i \text{ for } 1 \leq i < n$$
$$y_n = x_n \oplus x_1 x_2 .....x_{n-1}$$

An $n \times n$ Fredkin gate passes the first $n-2$ inputs to the output unchanged and swaps the last two inputs if the first $n-2$ inputs are 0. We will not be using the Fredkin gate in our synthesis algorithm.

Any Boolean function can be described as an XOR sum of products. The *positive-polarity Reed-Muller (PPRM)* expansion uses only uncomplemented variables and can be derived easily from the function's sum of products expansion. The PPRM of a function is unique and of the form:
$f(x_1, x_2, ..., x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus ... \oplus a_n x_n \oplus a_{12} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus ... \oplus a_{n-1,n} x_{n-1} x_n \oplus ... \oplus a_{12...n} x_1 x_2 ... x_n$, where $a_i \in \{0, 1\}$ and x_i are all uncomplemented (positive polarity).

III. THE SYNTHESIS ALGORITHM

Fig. 1 gives the main steps of the synthesis algorithm. In the first stage of the algorithm, a Reed-Muller expansion of all the output variables $v_{out,i}$ of the reversible function f is obtained in terms of all the input variables v_j and inserted into the priority queue. In the second stage, the algorithm enters a loop where it pops a node, *curnode*, from the priority queue. For each output variable, $v_{out,i}$, in *curnode*, the algorithm identifies factors, *fac*, in the PPRM expansion of $v_{out,i}$. For each factor, *fac*, identified in $v_{out,i}$, we make a copy of *curnode* in *newnode* and perform the substitution $v_i = v_i \oplus fac$ in the expansions of all the variables contained in *newnode*. Next, we examine *newnode*. If the synthesis is complete and the solution found is better than any previous solution, we cache *newnode*. Otherwise, we decide whether *newnode* presents an attractive path to follow for synthesis.

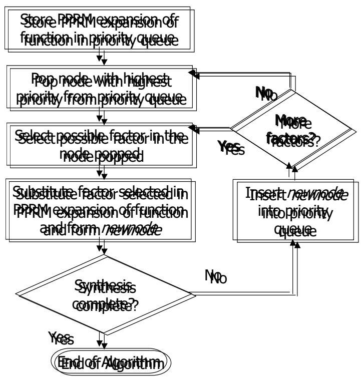
Store PPRM expansion of function in priority queue
Pop node with highest priority from priority queue
Select possible factor in the node popped
Substitute factor selected in PPRM expansion of function and form *newnode*
Synthesis complete?
Yes
End of Algorithm
No
Insert *newnode* into priority queue
More factors?
Yes
No

Fig. 1. Synthesis methodology

Depending on the decision, we insert *newnode* into the priority queue with a priority of: $newnode\_priority = \alpha * newnode.depth() + \beta * (init\_count - newnode.term\_count)/newnode.depth() - \gamma * (number\ of\ literals(fac))$. The first term gives preference to nodes of larger depth as all things being equal they are more likely to be close to the solution. The second term addresses the primary objective of minimizing the number of gates. The average number of terms eliminated per stage is used to measure a node's effectiveness. The third term addresses the secondary objective of minimizing the size of individual gates. The weights α, β, and γ add up to 1 and typical values were 0.2, 0.7, and 0.1, respectively. The algorithm continues in this manner until the time limit or the search tree is exhausted. Fig. 2 shows the search tree when the function to be synthesized is {1,0,7,2,3,4,5,6}, i.e., input vector 0 should map to output vector 1, input vector 1 to output vector 0, and so on.

IV. EXPERIMENTAL RESULTS

Table I depicts the results of applying various algorithms to all the 40,320 reversible functions of three variables. It shows how many reversible functions of three variables require a given number of gates. Synthesis only takes a total of few minutes on a 1.2GHz Athlon processor with 512 MB RAM for all these functions using the tool, called RMRLS, that implements our

Acknowledgments: This work was supported in part by NSF under grant No. CCR-0303789.

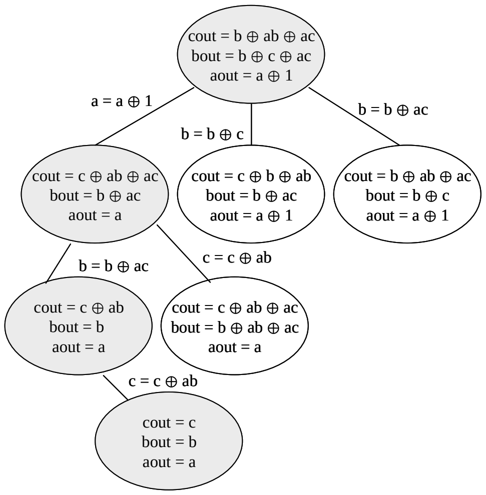

Fig. 2. Algorithm search tree

TABLE I
All reversible functions of three variables

#gates	Ours	Miller [5]	Optimal [4]
11		5	
10		110	
9	30	729	
8	3297	4726	577
7	12488	11199	10253
6	13620	12076	17049
5	7503	7518	8921
4	2642	2981	2780
3	625	767	625
2	102	130	102
1	12	15	12
0	1	1	1
Ave. size	6.10	6.18	5.87

synthesis methodology. As can be seen, our tool produces a lower average gate count than Miller's heuristic [5] and does not produce any circuits containing 10 or 11 Toffoli gates. Furthermore, our results compare very favorably to the optimal implementations found using depth-first search with iterative deepening (this is an exhaustive approach). Note that Miller's heuristic uses swap gates in addition to Toffoli gates, and is hence able to do better than the optimal method (which only uses Toffoli gates) for small gate counts. Thus, if our and the optimal methodologies are extended to handle swap gates in addition to Toffoli gates, their average gate counts would be even better than Miller's heuristic.

We now provide results for several examples from the literature. Unless otherwise stated, our results are equal in the number of gates to the best published solution and were synthesized in less than 0.1 seconds. The following examples are from [5] and [9].

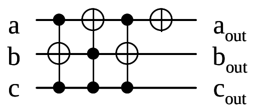

Fig. 3. Example 1 realization

Example 1: *Specification:* $\{1, 0, 3, 2, 5, 7, 4, 6\}$. *Toffoli network produced: TOF3(c,a,b) TOF3(c,b,a) TOF3(c,a,b) TOF1(a).* Thus there are four cascaded Toffoli gates as shown in Fig. 3.
Example 2: *Specification:* $\{7, 0, 1, 2, 3, 4, 5, 6\}$. *Toffoli network produced: TOF1(a) TOF2(a,b) TOF3(b,a,c).* The heuristic in [5] initially synthesized a circuit with seven gates for this specification which was improved to three gates using bidirectional synthesis.
Example 3: This example deals with the realization of a Fredkin gate using Toffoli gates. *Specification:* $\{0, 1, 2, 3, 4, 6, 5, 7\}$. *Toffoli network produced: TOF3(c,a,b) TOF3(c,b,a) TOF3(c,a,b).*
Example 4: Simple swap between two positions in the truth table. *Specification:* $\{0, 1, 2, 3, 4, 5, 6, 8, 7, 9, 10, 11, 12, 13, 14, 15\}$. *Toffoli network produced: TOF2(d,b) TOF3(d,b,a) TOF4(d,b,a,c) TOF4(c,b,a,d) TOF4(d,b,a,c) TOF3(d,b,a) TOF2(d,b).*
Example 5: This represents a wraparound shift of one position for four variables. *Specification:* $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0\}$. *Toffoli network produced: TOF4(c,b,a,d) TOF3(b,a,c) TOF2(a,b) TOF1(a).*

The above examples show that our algorithm matches the best results presented in the literature for specifications with a small number of variables. Due to the inability of previous algorithms to deal with specifications with a large number of variables, such examples do not exist in the literature. However, the main advantage of our algorithm is that it can easily tackle larger functions because of its ability to backtrack quickly once a path through the search tree has become inefficient. We next consider larger benchmarks.
Adder: This reversible full-adder generates sum, carry, and propagate signals, and requires one garbage output and one corresponding input for a total of four inputs and four outputs. *Specification:* $d_{out} = d \oplus ab \oplus bc \oplus ac$ (carry bit), $c_{out} = a \oplus b \oplus c$ (sum bit), $b_{out} = a \oplus b$ (propagate bit), and $a_{out} = a$ (garbage bit). *Toffoli network produced: TOF3(b,a,d) TOF2(a,b) TOF3(c,b,d) TOF2(b,c).*
rd53: Benchmark rd53 is from the MCNC [10] benchmark suite and we used the same reversible specification as in [5]. *Toffoli network produced: TOF2(c,b) TOF5(e,d,b,a,g) TOF3(b,a,f) TOF3(c,a,f) TOF2(b,a) TOF3(d,a,f) TOF2(a,d) TOF4(e,d,c,g) TOF3(e,d,f) TOF2(d,e) TOF5(d,c,b,a,g) TOF3(b,a,g).*
Shifter: This function has two control bits and ten input bits. It does a wraparound shift of zero, one, two, or three positions on the input depending on the control bits. *Toffoli network produced: a 26 gate solution was synthesized in 1.2 seconds.*

V. Conclusions

We have presented an algorithm which uses a positive-polarity Reed-Muller decomposition of a reversible function to select successive Toffoli gates. The algorithm searches the tree of possible factors in priority order to try to find the best possible solutions. We applied our algorithm to all 40,320 functions of three variables and obtained near-optimal results. Several examples of functions with a large number of variables were also presented to demonstrate the suitability of the algorithm for synthesizing complex functions. As part of future work, we would like to incorporate Fredkin gates into our algorithm. A Fredkin gate is equivalent to three Toffoli gates. Thus, the use of Fredkin gates could yield a significant improvement in circuit quality. We are also working on ways to efficiently synthesize functions with *don't cares* by using dynamic assignment instead of pre-assignment of values.

References

[1] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
[2] R. C. Merkle, "Two types of mechanical reversible logic," *Nanotechnology*, vol. 4, pp. 114–131, 1993.
[3] C. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev.*, vol. 17, pp. 525–532, 1973.
[4] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Reversible logic circuit synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2002, pp. 125–132.
[5] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proc. Design Automation Conf.*, June 2003, pp. 318–323.
[6] A. Khlopotine, M. Perkowski, and P. Kerntopf, "Reversible logic synthesis by iterative compositions," in *Proc. Int. Wkshp. Logic Synthesis*, June 2002, pp. 261–266.
[7] K. Iwama, Y. Kambayashi, and S. Yamashita, "Transformation rules for designing CNOT-based quantum circuits," in *Proc. Design Automation Conf.*, June 2002, pp. 419–424.
[8] A. Mishchenko and M. Perkowski, "Logic synthesis of reversible wave cascades," in *Proc. Int. Wkshp. Logic Synthesis*, June 2002, pp. 197–202.
[9] D. M. Miller and G. W. Dueck, "Spectral techniques for reversible logic synthesis," in *Proc. 6th Int. Symp. Representations & Methodology of Future Computing Technologies*, Mar. 2003.
[10] N.C.S.U. Collaborative Benchmarking Laboratory, Dept. of Computer Science, North Carolina State University, "Benchmark archives at CBL." [Online]. Available: http://www.cbl.ncsu.edu/benchmarks/