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 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 \( \{0,1,\ldots,2^n-1\} \) onto itself. An irreversible function can be converted into a reversible function easily. If the maximum number of identical output vectors is \( p \), then \( \log_2(p) \) 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. A \( n \times n \) Toffoli gate, denoted by \( TOF_n(x_1, x_2, \ldots, x_n) \), 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 \( 1 \times 1 \) Toffoli gate simply inverts the input unconditionally. Using \( x \) for input and \( y \) for output:

\[
y_i = x_i \quad \text{for} \quad 1 \leq i < n
\]

\[
y_n = x_n \oplus x_1 x_2 \cdots x_{n-1}
\]

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:

\[
f(x_1, x_2, \ldots, x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_{2^n-1} x_n \oplus a_{2^n} x_1 x_2 \oplus \cdots \oplus a_{2^n-1} x_{n-1} x_n \oplus \cdots \oplus a_{2^n-2} x_1 x_2 \cdots x_{n-1}
\]

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 \( v_i \) and inserted into the priority queue. In the second stage, the algorithm enters a loop where it pops a node, \( newnode \), from the priority queue. For each output variable, \( v_{out,i} \), in \( newnode \), the algorithm identifies factors, \( fac_i \), in the PPRM expansion of \( v_{out,i} \). For each factor, \( fac_i \), identified in \( v_{out,i} \), we make a copy of \( newnode \) in \( newnode \) and perform the substitution \( v_i = v_i \oplus fac_i \) in 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.

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 takes a total of a 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.

1530-1591/04 $20.00 (c) 2004 IEEE
than 0.1 seconds. The following examples are from [5] and [9].

Example 1: Specification: \{1, 0, 2, 3, 4, 6, 5, 7\}. Toffoli network produced: TOF3(c,a,b) TOF3(c,b,a) TOF3(c,b,a) 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, 5, 7\}. Toffoli network produced: TOF3(c,a,b) TOF3(c,b,a) TOF3(c,b,a) TOF1(a).

We now provide results for several examples from the literature.

### Table 1: All Reversible Functions of Three Variables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>5</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>729</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3297</td>
<td>4726</td>
<td>577</td>
</tr>
<tr>
<td>8</td>
<td>12488</td>
<td>11199</td>
<td>10253</td>
</tr>
<tr>
<td>7</td>
<td>13620</td>
<td>12076</td>
<td>17049</td>
</tr>
<tr>
<td>6</td>
<td>7503</td>
<td>7518</td>
<td>8921</td>
</tr>
<tr>
<td>5</td>
<td>2642</td>
<td>2981</td>
<td>2780</td>
</tr>
<tr>
<td>4</td>
<td>625</td>
<td>767</td>
<td>625</td>
</tr>
<tr>
<td>3</td>
<td>102</td>
<td>130</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ave. size 6.10 6.18 5.87

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


