Reversible Logic Circuit Synthesis

Vivek V. Shende, Aditya K. Prasad, Igor L. Markov and John P. Hayes
University of Michigan
Outline

- Motivation
  - Real-world Applications
  - Theoretical Advantages
  - Links to Quantum Computation
- Background
- Theoretical Results
- Synthesis of Optimal Circuits
- An Application to Quantum Computing
Real-world Applications

- Many inherently reversible applications
- Info. is re-coded, but none is lost or added
  - Digital signal processing
  - Cryptography
  - Communications
  - Computer graphics
  - Network congestion modeling
Theoretical Advantages

- Information conservation laws in physics
  - Thermodynamics ties irreversibility to dissipated heat: every lost bit causes an energy loss
  - C. Bennett, 1973, *IBM J. of R & D*

- Energy-lossless circuits ($Time \rightarrow \infty$)
  - must be information-lossless
  - have been built: S. Younis and T. Knight, 1994, *Workshop on Low Power Design*
Links to Quantum Computation

- Quantum operations are all reversible
- Every (classical) reversible circuit may be implemented in quantum technology, with overhead
  - “Pseudo-classical” subroutines of quantum algos
  - Can be implemented in classical reversible logic circuits
Outline

- Motivation
- Background
  - Reversibility
  - Permutations
  - Known Facts
- Theoretical Results
- Synthesis of Optimal Circuits
- An Application to Quantum Computing
Reversibility in Logic Gates

- **Definition**: reversible logic gate
  - \#input wires = \#output wires
  - Permutes the set of input values

- **Examples**
  - Inverter
  - 2-input, 2-output SWAP (S) gate
  - $k$-CNOT gate
    - $(k+1)$-inputs and $(k+1)$-outputs
    - Values on the first $k$ wires are unchanged
    - The last value is flipped if the first $k$ were all 1
Reversibility in Logic Circuits

- **Definition:**
  A combinational logic circuit is reversible iff
  - It contains only reversible gates
  - It has no fan-out
  - It is acyclic (as a directed multi-graph)

- **Theorem:**
  A reversible circuit must
  - Have as many input wires as output wires
  - Permute the set of input values
Example: A Reversible Circuit and Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a'</th>
<th>b'</th>
<th>c'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Equivalent to a NOT gate on wire c
Circuit Equivalences

Figure 2: Two reversible circuit equivalences. \( T(1,2;3) \cdot N(1) \cdot T(1,2;3) \cdot N(1) = C(2;3), \) and \( C(3;2) \cdot C(2;3) \cdot C(3;2) = S(2,3) \)

- Circuit equivalences: useful in synthesis
- More will be shown later
Reversible Circuits & Permutations

- A reversible gate (or circuit) with \( n \) inputs and \( n \) outputs has
  - \( 2^n \) possible input values
  - \( 2^n \) possible output values
- The function it computes on this set must, by definition, be a permutation
- The set of such permutations is called \( S_{2n} \)
Basic Facts About Permutations

- Permutations are multiplied by first applying one, then the other
  - example: \((1,2) \circ (2,3) = (1,3,2)\)
- A transposition
  - permutes exactly two elements
  - does not change any others
- Every permutation can be written as a product of transpositions
Even Permutations

- Consider all possible decompositions of a permutation into transpositions
- **Theorem**: The parity of the number of transpositions is constant
- **Definition**: Even permutations are those for which the number of transpositions is even
Known Facts

- **Fact 1**: Consider a reversible circuit
  - $n+1$ inputs and $n+1$ outputs
  - Built from gates which have at most $n$ inputs and $n$ outputs
  - Must compute an even permutation

- **Fact 2**: A universal gate library
  - CNOT, NOT, and TOFFOLI ("CNT")
  - Temporary storage may be required
Temporary Storage

Figure 3: A reversible circuit with $n-k$ wires of temp. storage.
Outline

- Motivation
- Background
- Theoretical Results
  - Zero-storage Circuits
  - Reversible De Morgan’s Laws
- Synthesis of Optimal Circuits
- An Application to Quantum Computing
Minimizing Temporary Storage

- Consider CNT circuits
  - **Theorem**: even permutations computable by circuits without temporary storage
  - **Theorem**: odd permutations computable with one line of temporary storage
- Same holds for NT and CNTS circuits
- The proof is constructive and may be used as a synthesis heuristic
Outline of Proof

- Explicitly construct a circuit to compute an arbitrary pair of disjoint transpositions \((A, B) (C, D)\) is okay; \((A, B) (B, C)\) is not
- Pick an even permutation
- Decompose it into transpositions
  - Will have an even number of transpositions
- Pair these up, guaranteeing disjointness
- Apply above construction to each pair
Reversible De Morgan’s Laws (1)

- De Morgan’s Laws
  - Apply to AND/OR/NOT circuits
  - Allow pushing all inverters to the inputs
- Reversible De Morgan’s Laws
  - Applied to reversible CNT circuits
  - Allow pushing all inverters to the inputs
- Pictures follow
Reversible De Morgan’s Laws (2)

- Similar rules exist for interchanging TOFFOLI and CNOT gates
- However, it is **not** always possible to push all CNOT gates to the inputs
- Oddly enough, all CNOT gates can be pushed to the “middle” of the circuit
Reversible De Morgan’s Laws (3)
Reversible De Morgan’s Laws (4)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram 1" /></td>
<td><img src="image2.png" alt="Diagram 2" /></td>
<td><img src="image3.png" alt="Diagram 3" /></td>
<td><img src="image4.png" alt="Diagram 4" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Diagram 5" /></td>
<td><img src="image6.png" alt="Diagram 6" /></td>
<td><img src="image7.png" alt="Diagram 7" /></td>
<td><img src="image8.png" alt="Diagram 8" /></td>
</tr>
</tbody>
</table>

IWLS 2002
Outline

- Motivation
- Background
- Theoretical Results
- Synthesis of Optimal Circuits
  - Optimality
  - IDA* Search Algorithm
  - Circuit Libraries
- An Application to Quantum Computing
Optimality

- The cost of a circuit is its gate count
  - Other cost functions can be considered
- **Definition**: optimal reversible circuit
  - no circuit with fewer gates computes the same permutation
- **Theorem**: a sub-circuit of an optimal circuit is optimal
  - Proof: otherwise, can improve the sub-circuit
IDA* Search

- Checks all possible circuits of cost 1, then all possible circuits of cost 2, etc…
- Avoids the memory blowup of BFS
- Still finds optimal solutions (unlike DFS)
- Checking circuits of cost less than $n$
  - Is much faster than processing cost-$n$ circuits
Dynamic Prog + Circuit Libraries

- IDA* search requires a subroutine to check all circuits of cost $n$, for arbitrary $n$
  - Called iteratively for $1 \ldots n$
- Only need to check locally optimal circuits
- Build optimal circuit library bottom up by DP
  - Index optimal circuits by computed permutation
  - In practice use hash_map datastructure from STL
Synthesis Algorithm (1)

CIRCUIT synthesize(PERM)
{
  if (PERM==IDENTITY) return EMPTY_CCT;
  // otherwise, use IDA* to find a circuit.
  for(DEPTH ← 1, DEPTH < MAX DEPTH; DEPTH++)
  {
    CIRCUIT ← find_circ(DEPTH, PERM, EMPTY_CCT);
    if (CIRCUIT != NIL) return CIRCUIT;
  }
}
CIRCUIT find_circ(COST, PERM, CURR_CCT) {
    if (COST ≤ k) // if PERM can be computed by a circuit
        // with fewer at most k gates,
        // such a circuit must be in the library
        return CURR_CCT + LIB[DEPTH].find(PERM));
    else // The goal circuit must have >k gates;
        // Try constructing it from k-gate circuits
        for each C in LIB[k] {
            // divide PERM by permutation computed by C
            PERM2 ← PERM * INVERSE(C.perm)
            // and try to synthesize the result
            TEMP_CCT ← find_circ(depth-k,PERM2));
            if (TEMP_CCT != NIL) return TEMP_CCT;
        }
Empirical Circuit Synthesis

- Consider all reversible functions on 3 wires
  \(8! = 40,320\) functions
- For each gate library from \(N, C, T, NC, CT, NT, CNT, CNTS\)
  - Is it universal?
  - How many functions can it synthesize?
  - What are largest optimal circuits?
  - How long does it take to synthesize circuits?
# Optimal Circuit Sizes

<table>
<thead>
<tr>
<th>Size</th>
<th>N</th>
<th>C</th>
<th>T</th>
<th>NC</th>
<th>CT</th>
<th>NT</th>
<th>CNT</th>
<th>CNTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1690</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8363</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12237</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>9339</td>
<td>577</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>386</td>
<td>5097</td>
<td>10253</td>
<td>6817</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>215</td>
<td>1688</td>
<td>2262</td>
<td>17049</td>
<td>17531</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>474</td>
<td>1784</td>
<td>870</td>
<td>8921</td>
<td>11194</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>60</td>
<td>5</td>
<td>393</td>
<td>845</td>
<td>296</td>
<td>2780</td>
<td>3752</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>51</td>
<td>9</td>
<td>187</td>
<td>261</td>
<td>88</td>
<td>625</td>
<td>844</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>24</td>
<td>6</td>
<td>51</td>
<td>60</td>
<td>24</td>
<td>102</td>
<td>135</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>168</td>
<td>24</td>
<td>1344</td>
<td>5040</td>
<td>40320</td>
<td>40320</td>
<td>40320</td>
</tr>
<tr>
<td>Time, s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>215</td>
<td>97</td>
<td>40</td>
<td>15</td>
</tr>
</tbody>
</table>

*Michigan Engineering*  
IWLS 2002
Outline

- Motivation
- Background
- Theoretical Results
- Synthesis of Optimal Circuits
- An Application to Quantum Computing
  - Grover’s Search
  - Pseudo-classical Synthesis
Grover’s Search

- A quantum algorithm for associative search (input is not sorted)
  - Search criterion: a classical one-output function $f$
  - M. Nielsen and I. Chuang, 2000
- Runs in time $O(\sqrt{N})$
  - any classical algorithm provably requires $\Omega(N)$ time
- Requires a subroutine (oracle) that
  - changes the phase (sign) of all basis states (bit-strings) that match the search criterion $f$
Pseudo-classical Synthesis

- We focus on circuits for Grover oracles
- To change the sign of a bit-string
  - Initialize a qubit to $|0\rangle - |1\rangle$
  - Compute the classical one-output function $f$
  - XOR the qubit with $f$
  - Whenever $f=1$, the sign (phase) will change
- Thus, the design of Grover search circuits for a given $f$
  - Is reduced to reversible synthesis
  - Can be solved optimally by our methods
ROM-based Circuits

- Desired circuits must alter phase of basis states
  - All bits except one must be restored to input values
- Previous work studied ROM-based circuits
  - Constraint: ROM qubits can never change
  - Theorems + heuristic synthesis algorithms
- Our work: synthesis of pseudo-classical circuits
  - 3 read-only “ROM” wires that can never change
  - 1 wire that can be changed during computation, but must be restored by end
  - 1 wire on which function is computed
Synthesis Algorithms Compared

- Heuristic synthesis of ROM-based circuits
  - Proposed by Travaglione et al, 2001
  - Based on EXOR-sum decomposition (“EXOR”)
    (Our code uses Mishchenko’s EXORCISM-4)
  - Imposed a restriction: at most one control bit per gate can be on a ROM bit

- Optimal synthesis (as described earlier)
  - with restriction from Travaglione (“Opt T”)
  - without this restriction (“Opt”)
## Sizes of 3+2 ROM-circuits

<table>
<thead>
<tr>
<th>Size</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXOR</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>18</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>19</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>OptT</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>21</td>
<td>24</td>
<td>29</td>
<td>33</td>
<td>44</td>
<td>46</td>
<td>22</td>
</tr>
<tr>
<td>Opt</td>
<td>1</td>
<td>7</td>
<td>21</td>
<td>35</td>
<td>36</td>
<td>28</td>
<td>28</td>
<td>36</td>
<td>35</td>
<td>21</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXOR</td>
<td>8</td>
<td>10</td>
<td>16</td>
<td>19</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>12</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>OptT</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Opt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Discussion of Empirical Results

- The EXOR-SUM heuristic is sub-optimal
- All methods able to synthesize all 256 fns
  - “Opt T” able to synthesize as many as “Opt”:
    - B. Travaglione et al., 2001
- “Opt” results symmetrical about 5-6 gates
  - Function \( x \) requires one fewer gate than \( 256-x \)
  - Explanation yet to be found
- “Exor” results symmetrical about 13 gates
Conclusions

- Classical reversible circuits as special-case quantum circuits
- Existence theorems
- Reversible De Morgan’s laws
  - Future research on optimization heuristics
- Algorithm for synthesis of optimal circuits
  - Applicable to Grover’s search
Thank You For Your Attention