Reversible Logic Circuit Synthesis *

Vivek V. Shende, Aditya K. Prasad, Igor L. Markov, and John P. Hayes
Advanced Computer Architecture Laboratory, University of Michigan, Ann Arbor, MI 48109-2122
{vshende,akprasad,imarkov,jhayes}@umich.edu

ABSTRACT

Reversible or information-lossless circuits have applications in digital signal processing, communication, computer graphics and cryptography. They are also a fundamental requirement in the emerging field of quantum computation. We investigate the synthesis of reversible circuits that employ a minimum number of gates and contain no redundant input-output line-pairs (temporary storage channels). We prove constructively that every even permutation can be implemented without temporary storage using NOT, CNOT and TOFFOLI gates. We describe an algorithm for the synthesis of optimal circuits and study the reversible functions on three wires, reporting distributions of circuit sizes. Finally, in an application important to quantum computing, we synthesize oracle circuits for Grover's search algorithm, and show a significant improvement over a previously proposed synthesis algorithm.

1. INTRODUCTION

In most computing tasks, the number of output bits is relatively small compared to the number of input bits. For example, in a decision problem, the output is only one bit (yes or no) and the input can be as large as desired. However, computational tasks in digital signal processing, communication, computer graphics and cryptography require that all of the information encoded in the input be preserved in the output. Some of those tasks are important enough to justify adding new microprocessor instructions to the HP PA-RISC (MAX and MAX-2), Sun SPARC (VIS), PowerPC (AltiVec), IA-32 and IA-64 (MMX) instruction sets [13, 8]. In particular, new bit-permutation instructions were shown to vastly improve performance of several standard algorithms, including matrix transposition and DES, as well as two recent cryptographic algorithms Twofish and Serpent [8]. Bit permutations are a special case of *reversible functions*, that is, functions that permute the set of possible input values. For example, the butterfly operation $(x,y) \rightarrow (x+y, x-y)$ is reversible but is not a bit permutation. It is a key element of Fast Fourier Transform algorithms and has been used in application-specific processors from Tensilica. One might expect to get further speed-ups by adding instructions to allow computation of an arbitrary reversible function. The problem of chaining such instructions together provides one motivation for studying reversible logic circuits, that is, logic circuits composed of gates computing reversible functions.

Reversible circuits are also interesting because the loss of information implies energy loss [2]. Younis and Knight [16] showed that some reversible circuits can be made asymptotically energy-lossless if their delay is allowed to be arbitrarily large. Currently, energy losses due to irreversibility are dwarfed by the overall power dissipation, but this may change if power dissipation improves. In particular, reversibility is important for nanotechnologies where switching devices with gain are difficult to build.

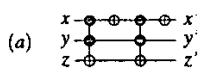
Finally, reversible circuits can be viewed as a special case of quantum circuits because quantum evolution must be reversible [9]. Classical (non-quantum) reversible gates are subject to the same "circuit rules", whether they operate on classical bits or quantum states. In fact, popular universal gate libraries for quantum computation often contain as subsets universal gate libraries for classical reversible computation. While the speed-ups which make quantum computing attractive are not available without purely quantum gates, logic synthesis for classical reversible circuits is a first step toward synthesis of quantum circuits. Moreover, algorithms for quantum communications and cryptography often do not have classical counterparts because they act on quantum states, even if their action in a given computational basis corresponds to classical reversible functions on bit-strings. Another connection between classical and quantum computing comes from Grover's search algorithm. Circuits for Grover's algorithm contain large parts consisting of NOT, CNOT and TOFFOLI gates only [9].

We review existing work on classical reversible circuits [10]. Toffoli [14] gives constructions for an arbitrary reversible or irreversible function in terms of a certain gate library. However, his method makes use of a large number of temporary storage channels, i.e. input-output wire-pairs other than those on which the function is computed. Sasao and Kinoshita show that any conservative function ($f(x)$ is conservative if for all x, x and $f(x)$ contain the same number of 1s in their binary expansions) has an implementation with only three temporary storage channels using a certain fixed library of conservative gates, although no explicit construction is given [11]. Kerntopf uses exhaustive search methods to examine small-scale synthesis problems and related theoretical questions about reversible circuit synthesis [5].

Our work pursues synthesis of optimal reversible circuits which can be implemented without temporary storage channels. In Section 3 we show by explicit construction that any reversible function which performs an even permutation on the input values can be synthesized using the CNTS (CNOT, NOT, TOFFOLI, and SWAP) gate library under such constraints. In Section 4 we present synthesis algorithms for decomposing such a function into a circuit with a minimal number of gates. Besides branch-and-bound, we use a dynamic programming technique that exploits reversibility. Applications to quantum computing are examined in Section 5.

*This work was partially supported by the Undergraduate Summer Research Program at the University of Michigan and by the DARPA QuIST program. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies of endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

ICCAD 2002 Nov 10-14, 2002, San Jose, California, USA

(a) 

(b)
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

(c)

x	y	z	x'	y'	z'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figure 1: (a) A 3×3 reversible circuit computing CNOT, (b) the corresponding matrix, and (c) its truth table.

2. BACKGROUND

In conventional (irreversible) circuit synthesis, one typically starts with a universal gate library and some specification of a Boolean function. The goal is to find a logic circuit that implements the Boolean function and minimizes a given cost metric, e.g., the number of gates or the circuit depth. At a high level, reversible circuit synthesis is just a special case in which no fanout is allowed and all gates must be reversible.

DEFINITION 1. *A gate is reversible if the (Boolean) function it computes is bijective.*

A necessary condition is that the gate have the same number of input and output wires. If it has k, it is called a $k \times k$ gate, or a gate on k wires. We will think of the mth input wire and the mth output wire as really being the same wire. Many gates satisfying these conditions have been examined. We will consider a specific set defined by Toffoli [14].

DEFINITION 2. *A k-CNOT is a $(k+1) \times (k+1)$ gate. It leaves the first k inputs unchanged, and inverts the last iff all others are* 1.

Clearly the k-CNOT gates are all reversible. The first three of these have special names. The 0-CNOT is just an inverter, referred to as a NOT gate, and denoted N. It performs the operation $(x) \rightarrow (x \oplus 1)$, where $\oplus$ denotes XOR. The 1-CNOT, which performs the operation $(y,x) \rightarrow (y, x \oplus y)$ is referred to as a Controlled-NOT, or CNOT (C). The 2-CNOT is called a TOFFOLI (T) gate, and performs the operation $(z,y,x) \rightarrow (z,y,x \oplus yz)$. We will also be using another reversible gate, called the SWAP (S) gate. It is a 2×2 gate which exchanges the inputs; that is, $(x,y) \rightarrow (y,x)$. One reason for choosing these particular gates is that they appear often in the quantum computing context [9]. We will be working with circuits from a given, limited-gate library. Usually, this will be the CNTS gate library, consisting of the CNOT, NOT, and TOFFOLI, and SWAP gates defined above.

DEFINITION 3. *A well-formed reversible logic circuit is an acyclic combinational logic circuit in which all gates are reversible, and are interconnected without fanout.*

As with reversible gates, a reversible circuit has the same number of input and output wires; again we will call a reversible circuit with n inputs an $n \times n$ circuit, or a circuit on n wires. We can also think of an $n \times n$ circuit as the inner workings of an $n \times n$ reversible gate.

This also allows us to draw reversible circuits as arrays of horizontal lines representing wires, in which gates are represented by vertically-oriented symbols. For example, in Figure 1*a*, we see a reversible circuit drawn in standard notation [9]. The $\oplus$ symbols represent inverters and the $\bullet$ symbols represent controls. A vertical line connecting a control to an inverter means that the inverter is only applied if the wire on which the control is set carries a 1 signal. Thus, the gates used are, from left to right, TOFFOLI, NOT, TOFFOLI, and NOT.

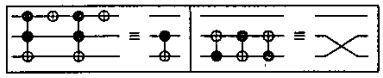


Figure 2: Reversible circuit equivalences: (a) T(1,2;3) · N(1) · T(1,2;3) · N(1) = C(2;3), and (b) C(3;2) · C(2;3) · C(3;2) = S(2,3).

Since we will be dealing only with bijective functions, we represent them using the *cycle notation*, from elementary algebra, where a permutation is represented by disjoint cycles of variables. For example, the truth table in Figure 1*b* is represented by $(2,3)(6,7)$ because the corresponding function swaps 010 (2) and 011 (3), and 110 (6) and 111 (7). The set of all permutation of n marks is denoted S_n, so the set of bijective functions with n binary inputs is S_{2^n}. We will call $(2,3)(6,7)$ *CNT-constructible* since it can be computed by a circuit with gates from the CNT gate library. More generally:

DEFINITION 4. *Let L be a (reversible) gate library. An L-circuit is a circuit with only gates from L. A permutation $\pi \in S_{2^n}$ is L-constructible if it can be computed by an $n \times n$ L-circuit.*

In Figure 2*a* we see that the circuit in Figure 1*a* is equivalent to one consisting of a single C gate. Pairs of circuits computing the same function are very useful, since we can substitute one for another. On the right, we see similarly that three C gates can be used to replace a S gate. Figure 2 therefore shows us that the C and S gates in the CNTS gate library can be removed without losing computational power. We will still use the CNTS gate library in synthesis to reduce gate counts and potentially speed up synthesis This is motivated by Figure 2, which shows how to replace four gates with one C gate, and thus up to 12 gates with one S.

DEFINITION 5. *Two reversible circuits are equivalent if they compute the same function.*

Figure 2*a* illustrates the use of "temporary storage". A C gate only needs two wires, but if we simulate it with two N gates and two T gates, we need a third wire. The value of the third wire emerges unaltered. More generally, consider the general reversible circuit of Figure 3. The top $n-k$ lines transfer $n-k$ signals Y to the corresponding wires on the other side of the circuit. The bottom k wires enter as the input value X and emerge as the output value $f(X)$. These wires usually serve as an essential workspace for computing $f(X)$. Following Toffoli, we say this circuit computes $f(X)$ using $n-k$ lines of temporary storage [14].

DEFINITION 6. *Let L be a reversible gate library. Then L is universal if for all k and all permutations $\pi \in S_{2^k}$, there exists some l such that some L-constructible circuit computes π using l wires of temporary storage. (Note that we do not assume that fixed inputs are available.)*

It is a result of Toffoli's that the CNT gate library is universal; he also showed that one can bound the amount of temporary storage

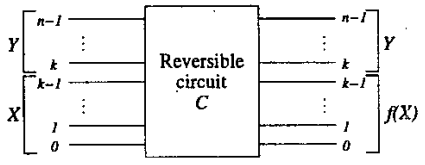


Figure 3: Circuit C with $n-k$ wires Y of temporary storage.

required to compute a permutation in S_{2^n} by $n-3$. We are interested in trying to synthesize permutations using no extra storage. For an example of what limitations this puts on the set of computable permutations, suppose we were working with only the C gate library. Then the following is true:

PROPOSITION 1. *Every C-constructible permutation computes an invertible linear transformation. Moreover, every invertible linear transformation is computable by a C-constructible circuit. Finally, S_{2^n} has $\prod_{i=0}^{n-1}(2^n-2^i)$ C-constructible permutations.*

Proof: A function f is linear iff $f(\mathbf{x}\oplus\mathbf{y}) = f(\mathbf{x})\oplus f(\mathbf{y})$, where $\oplus$ denotes bitwise XOR. The composition of two linear functions is a linear function. Therefore, to show that all C circuits are linear, it suffices to prove each C gate computes a linear transformation. Indeed, $C(x_1\oplus y_1, x_2\oplus y_2) = (x_1\oplus y_1, x_1\oplus y_1\oplus x_2\oplus y_2) = (x_1, x_1\oplus y_1)\oplus(x_2, x_2\oplus y_2) = C(x_1,y_1)\oplus C(x_2,y_2)$. On the other hand, observe that the linearity in terms of bit-wise $\oplus$ matches the linearity in vector spaces over the two-element field $\mathbb{F}_2$. In the basis $10\ldots0, 01\ldots0, \ldots, 0\ldots01$, the matrices corresponding to individual C gates account for all the elementary row-addition matrices. An example is given in Figure 1. Because there is only one non-zero scalar in $\mathbb{F}_2$, any invertible matrix in $GL(\mathbb{F}_2)$ can be written as a product of these. Thus, any invertible linear transformation can be computed by a C-circuit.

Finally, a linear mapping is fully defined by its values on basis vectors. There are 2^n-1 ways of mapping the 2^n-bit string 10...0. Once we fixed its image, there are 2^n-2 ways of mapping 010...0, and so on. Each time we map one of these basis bit-strings it can't map to the subspace spanned by the previous bit-strings. There are 2^n-2^i choices for the i-th basis bit-string. Once all basis bit-strings are mapped, the mapping of the rest is specified by linearity. □

A similar result for CNT-constructible permutations requires:

DEFINITION 7. *A permutation is called even if it can be written as the product of an even number of transpositions. The set of even permutations in S_n is denoted A_n.*

It is well-known that if a permutation can be written as the product of an even number of transpositions, then it may not be written as the product of an odd number of transpositions. Moreover, half the permutations in S_n are even for $n > 1$.

PROPOSITION 2. *Any $n\times n$ circuit with no $n\times n$ gates computes an even permutation* [14].

To illustrate this proposition, consider the following example. A 2×2 circuit consisting of a single S gate performs the permutation $(1,2)$, as the inputs 01 and 10 are interchanged, and the inputs 00 and 11 remain fixed. This permutation consists of one transposition, and is therefore odd. On the other hand, in a 3×3 circuit, one can check that a swap gate on the bottom two wires performs the permutation $(1,2)(5,6)$, which is even.

3. THEORETICAL RESULTS

Since the CNTS gate library contains no gates of size greater than three, Proposition 2 implies that every CNTS-constructible (without temporary storage) permutation is even for $n \geq 4$. The converse is true as well:

PROPOSITION 3. *Every even permutation is CNT-constructible.*

Proof: It follows from a result of Toffoli's [14] that every permutation in S_{2^n} is CNT-constructible for $n < 4$. This is explicitly verified in Table 1. Suppose $n \geq 4$. Any permutation $\pi \in A_{2^n}$ can be written as the product of pairs of disjoint transpositions; for a proof, see

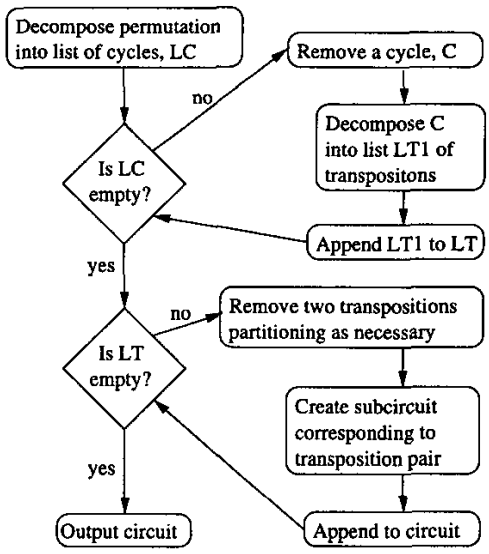


Figure 5: Flowchart for algorithm from proof of Proposition 3.

Proposition 13 in the Appendix. It therefore suffices to show that pairs of disjoint transpositions are constructible, as we can chain together their circuits to obtain the circuit for π. First, we observe that the permutation with cycle decomposition $(0,1)(2,3)$ can be computed by a circuit consisting of a $(n-2)$-CNOT gate with the controls on the top $n-2$ wires and the inverter on the bottom wire, and with an N gate on each side of each control. We can replace the $(n-2)$-CNOT gate with a $8(n-5)$ T gates [1, Corollary 7.4]. Let $S = \{a,b,c,d\}$ with a,b,c,d distinct. In Proposition 14 in the Appendix, we explicitly construct a circuit computing a permutation π_S such that $\pi_S(a) = 0$, $\pi_S(b) = 1$, $\pi_S(c) = 2$, and $\pi_S(d) = 3$. Because $(a,b)(c,d) = \pi_S(0,1)(2,3)\pi_S^{-1}$, we can sequence the circuits for π_S, $(0,1)(2,3)$, and π_S^{-1}, to obtain a circuit for $(a,b)(c,d)$. □

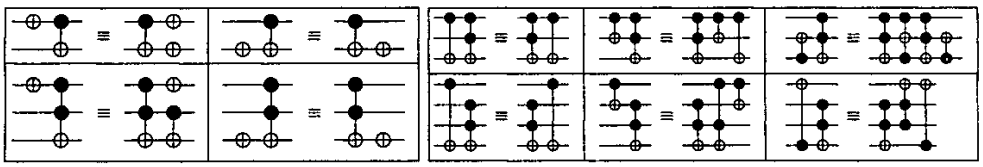
The following two corollaries give a way to synthesize circuits computing odd permutations using temporary storage, and also extend Proposition 3 to an arbitrary universal gate library.

PROPOSITION 4. *Every permutation is CNT-constructable with at most one wire of temporary storage.*

Proof: Suppose we have a $n\times n$ gate G computing $\pi \in S_{2^n}$, and we place it on the bottom n wires of an $(n+1)\times(n+1)$ reversible circuit; let $\bar{\pi}$ be the permutation computed by this new circuit. Then by Proposition 2, $\bar{\pi}$ is even. Another way of seeing is this to observe that each cycle in π appears "twice" in $\bar{\pi}$, once when the top wire carries 0 and once when it carries 1. By Proposition 3, $\bar{\pi}$ is CNT-constructible. Let C be a CNT-circuit computing $\bar{\pi}$. Then C computes π with one line of temporary storage. □

PROPOSITION 5. *For any universal gate library L and sufficiently large n, permutations in A_{2^n} are L-constructible, and those in S_{2^n} are realizable with at most one wire of temporary storage.*

Proof: Since L is universal, there is some number k such that we can compute the permutations corresponding to the NOT, CNOT, and TOFFOLI gates using k total wires. Let $n > k$, and let $\pi \in A_{2^n}$. By Proposition 3, we can find a CNT-circuit C computing π, and can replace every occurrence of N, C, or T gate with a circuit computing it. The second claim follows from Propositions 3 and 4. □



(a) (b)

Figure 4: Equivalences between reversible circuits used in our constructions.

Proposition 3 is proven by an explicit construction, which constitutes a (non-optimal) circuit synthesis heuristic; see Figure 5. For permutations in A_{2^n}, the runtime and the length of the circuits produced are both $\Theta(n2^n)$ in the worst case. In general, the complexity is $\Theta(ns)$ where s is the number of indices moved by the permutation we are trying to synthesize. This agrees with the above estimate, as at most 2^n indices may be moved.

Later, we describe an algorithm which synthesizes optimal circuits using an arbitrary gate library. Roughly speaking, the performance of this algorithm is improved by using a smaller gate library, as long as the average circuit length is not significantly increased.

DEFINITION 8. *For any gate libraries $L_1 \ldots L_k$, a $L_1|\ldots|L_k$-circuit is an L_1-circuit followed by an L_2-circuit, ..., followed by an L_k-circuit. A permutation computed by an $L_1|\ldots|L_k$-circuit is $L_1|\ldots|L_k$-constructible.*

PROPOSITION 6. *For every CNT-circuit there is an equivalent CT|N-circuit.*

Proof: First, we move all the N gates toward the outputs of the circuit. Each box in Figure 4*a* indicates a way of replacing an N|CT circuit with a CT|N circuit. Moreover, every possible way for an N gate to appear to the immediate left of a C or a T is accounted for, up to permuting the input and output wires. Now, number the non-N gates in the circuit in a reverse topological order starting from the outputs. In particular, if two gates appear at the same level in a circuit diagram, they must be independent, and one can order them arbitrarily. Let d be the number of the highest-numbered gate with an N gate to its left. All N gates past the d-th gate G can be reordered with the G gate without introducing new N gates on the other side of G. In any event, as there are no remaining N gates to the left of G anymore, d decreases. This process terminates with all the N gates are clustered together at the circuit outputs. If we always cancel redundant N gates, then no more than two new gates will be introduced for each non-inverter originally in the circuit; additionally, there will be no more than n total N gates when the process is complete. Thus if the original circuit had l gates, then the new circuit has at most $3(l-1)+n$ gates. □

PROPOSITION 7. *The permutation π computed by a CT|N-circuit determines the π_{CT} and π_N computed by the CT and N sub-circuits.*

Proof: C and T gates (and hence CT-circuits) fix 0. Thus $\pi(0) = \pi_N(0)$. But the image of 0 (or anything else) under an N-circuit completely determines the π_N. Hence $\pi_{CT} = \pi\pi_N^{-1} = \pi\pi_N$. □

Thus, if we want a CNT-circuit computing a permutation π, we can quickly compute π_N and then simplify the problem to that of finding a CT-circuit for $\pi\pi_N$. By Proposition 6, we know that a minimal-gate circuit of this form has at most about three times as many gates as the gate-minimal circuit computing π.

Figure 4*b* shows how to move a C gate past a T gate, and account for every possible way a C can appear to the left of a T, up to permuting wires. From this, one might expect every CT circuit to be equivalent to a T|C circuit. This is not the case, however. We note that the proof of Proposition 6 in fact requires the ability to move an arbitrary number of N gates past any other given gate, while Figure 4*b* only allows us to move one C gate past a given T gate. However, many CT circuits are equivalent to T|C circuits, and the following result holds:

PROPOSITION 8. *The permutation π computed by a T|C-circuit determines permutations π_T and π_C computed by the sub-circuits.*

Proof: By Proposition 1, any C-circuit is linear, so it suffices to check its values on the basis elements (binary expansions of 2^i). As any T circuit fixes these, $\pi(2^i) = \pi_C \circ \pi_T(2^i) = \pi_C(2^i)$, so the permutation π uniquely determines π_C. $\pi_T = \pi\pi_C^{-1}$. □

Proposition 8 implies that the number of T|C-constructible permutations in S_{2^k} is equal to the number that are C-constructible times the number that are T-constructible. In Section 4, we use this to show that there exist CT-constructible permutations which are not T|C-constructible.

4. OPTIMAL SYNTHESIS

Now that we know which permutations admit circuit realizations without extra storage, we seek *optimal* realizations of this type. A circuit is optimal if no equivalent circuit has smaller cost; in our case, the cost function will be the number of gates in the circuit.

PROPOSITION 9. *(Property of Optimality) If B is a sub-circuit of an optimal circuit A, then B is optimal.*

Proof: Suppose not. Then let B' be a circuit with fewer gates than B, but computing the same function. If we replace B by B', we get another circuit A' which computes the same function as A. But since we have only modified B, A' must be as much smaller than A as B' is smaller than B. However, A was assumed to be optimal, hence this is a contradiction. Note: equivalent, optimal circuits can have the same number of gates. □

Proposition 9 allows us to build a library of small optimal circuits by dynamic programming because the first m gates of an optimal $(m+1)$-gate circuit form an optimal subcircuit. Therefore, to examine all optimal $(m+1)$-gate circuits, we iterate through optimal m-gate circuits and add single gates at the end in all possible ways. Some of the $(m+1)$-gate circuits found may have been synthesized with fewer gates. Those which have not are optimal. In fact, instead of storing a library of all optimal circuits, we store one optimal circuit per synthesized permutation and also store optimal circuits of a given size together.

One way to find an optimal circuit for a given permutation π is to generate all optimal k-gate circuits for increasing values of k until a circuit computing π is found. This procedure requires $\Theta(2^n!)$ memory in the worst case (n is the number of wires) and may require more memory than is available. Therefore, we stop growing

```
CIRCUIT find_circ(COST, PERM, CURR_CCT)
// assumes circuit library stored in LIB

if (COST ≤ k)

  // If PERM can be computed by a circuit with ≤ k gates,
  // such a circuit must be in the library
    return CURR_CCT * LIB[DEPTH].find(PERM)

else

  // Try building the goal circuit 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
```

Figure 6: Finding a circuit of cost ≤COST that computes permutation PERM (NIL returned if no such circuit exists). CURR_CCT, TEMP_CCT and records in LIB represent circuits, and include a field "perm" storing the permutation computed. The * character means concatenation of circuits, and NIL*<anything>=NIL.

Size	N	C	T	NC	CT	NT	CNT	CNTS
12	0	0	0	0	0	47	0	0
11	0	0	0	0	0	1690	0	0
10	0	0	0	0	0	8363	0	0
9	0	0	0	0	0	12237	0	0
8	0	0	0	0	6	9339	577	32
7	0	0	0	14	386	5097	10253	6817
6	0	2	0	215	1688	2262	17049	17531
5	0	24	0	474	1784	870	8921	11194
4	0	60	5	393	845	296	2780	3752
3	1	51	9	187	261	88	625	844
2	3	24	6	51	60	24	102	134
1	3	6	3	9	9	6	12	15
0	1	1	1	1	1	1	1	1
Total	8	168	24	1344	5040	40320	40320	40320
Time	1	1	1	30	215	97	40	15

Table 1: Number of permutations computable in an optimal L-circuit using a given number of gates. $L \subset CNTS$. Runtimes are given in seconds for a 2GHz Pentium-4 Xeon workstation.

the circuit library at m-gate circuits, when hardware limitations become an issue. The second stage of the algorithm uses the computed library of optimal circuits and, in our implementation, starts by reading the library from a file. Since little additional memory is available, we trade off runtime for memory.

We use a technique known as *depth-first search with iterative deepening* (DFID) [6]. After a given permutation is checked against the circuit library, we seek circuits with $j = m+1$ gates that implement this permutation. If none are found, we seek circuits with $j = m+2$ gates, etc. This algorithm, in general, needs an additional termination condition to prevent infinite looping for inputs which cannot be synthesized with a given gate library. For each j, we consider all permutations optimally synthesizable in m gates. For each such permutation ρ, we multiply π by ρ^{-1} and recursively try to synthesize the result using $j-m$ gates. When $j-m \leq m$, this can be done by checking against the existing library. Otherwise, the recursion depth increases. Pseudocode for this stage of our algorithm is given in Figure 6.

In addition to being more memory-efficient than straightforward dynamic programming, our algorithm is faster than branching over all possible circuits. To quantify these improvements, consider a library of circuits of size m or less, containing l_m circuits of size m. We analyze the efficiency of the algorithms discussed by simulating them on an input permutation of cost k. Our algorithm requires $l_m^{\lfloor (k-1)/m \rfloor}$ references to the circuit library. Simple branching is no better than our algorithm with $m = 1$, and thus takes at least l_1^k steps, which is $l_1^k / l_m^{\lfloor (k-1)/m \rfloor}$ times more than our algorithm. A speed-up can be expected because $l_m \leq l_1^m$, but specific numerical values of that expression depend on the numbers of suboptimal and redundant optimal circuits of length m. Indeed, Table 1 lists values of l_m for various subsets of the CNTS gate library and $m = 3$. For example, for the NT gate gate library, $k = 12$, $\lfloor (k-1)/m \rfloor = 3$, $l_1 = 6$ and $l_m = 88$. Therefore the performance ratio is $l_1^k / l_m^{\lfloor (k-1)/m \rfloor} = 6^{12}/88^3 \approx 3194.2$. Yet, this comparison is incomplete because it does not account for time spent building circuit libraries. We point out, however, that this charge is amortized over multiple synthesis operations. In our experiments, generating a circuit library on three wires of up to three gates ($m = 3$) from the CNTS gate library takes less than a minute on a 2-GHz Pentium-4 Xeon. Using such libraries, all of Table 1 can be generated in minutes,[1] but cannot be generated even in several hours using branching.

Let us now see what additional information we can glean from Table 1. Adding the C gate to the NT library appears to significantly reduce circuit size, but further adding the S gate does not help as much. To illustrate this, we show sample worst-case circuits on three wires for the NT, CNT, and CNTS gate libraries in Figure 7.

The totals in Table 1, can be independently determined by the following arguments. Every reversible function on three wires can be synthesized using the CNT gate library [14], and there are $8! = 40,320$ of these. All can be synthesized with the NT library because the C gate is redundant in the CNT library; see Figure 2*a*. On the other hand, adding the S gate to the library cannot decrease the number of synthesizable functions. Therefore, the totals in the NT and CNTS columns must be 40,320 as well. On the other side of the table, the number of possible N circuits is just $2^3 = 8$ since there are three wires, and there can be at most one N gate per wire in an optimal circuit (else we can cancel redundant pairs.) By Propositions 6 and 7, the number of CN-constructible permutations should be the product of the number of N-constructible permutations and the number of C constructible permutations, since any CN-constructible permutation can be written uniquely as a product of an N-constructible and a C-constructible permutation. So the total in the CN column should be the product of the totals in the C and N columns, which it is. Similarly, the total in the CNT column should be the product of the totals in the CT and N columns; this allows one to deduce the total number of CT-constructible permutations from values we know. Finally, Proposition 1 states that the number of permutations implementable on n wires with C gates is $\prod_{i=0}^{n-1}(2^n - 2^i)$. For $n = 3$ this yields 168 and agrees with Table 1.

[1] Although complete statistics for all 16! 4-wire functions are beyond our reach, average synthesis times are less than one second when the input function can be implemented with eight gates or less. Functions requiring nine or more gates tend to take more than 1.5 hours to synthesize. In this case memory constraints limit our circuit library to 4-gate circuits, and the large jump in runtime after the 8-gate mark is due to an extra level of recursion.

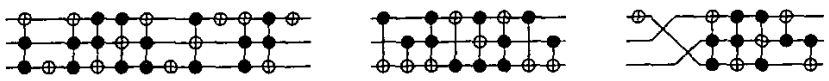


Figure 7: Worst-case L-circuits where L is NT, CNT and CNTS.

We can also add to the discussion of T|C constructible circuits we began in Section 3. By Proposition 7, the number of T|C-constructible permutations can be computed as the product of the numbers of T-constructible and C-constructible permutations. Table 1 mentions 24 T-circuits and 168 C-circuits on three wires. The product, 4032, is less than 5040, the number of CT constructible permutations on three wires. Therefore:

PROPOSITION 10. *There exist CT constructible permutations in S_8 which are not T|C constructible.*

Finally, we observe that the longest optimal C-circuits on 3, 4 and 5 wires merely permute the wires. Our experimental data supports the conjecture that no optimal C-circuit on n wires has more than $3(n-1)$ gates, and the ones with $3(n-1)$ gates represent wire permutations that leave no wire fixed. However, an information-theoretic counting argument shows that the optimal gate count in an optimal C-circuit is at least $O(n^2/log(n))$. This asymptotic bound is produced by comparing the number of unique C-circuits on n wires and the number of circuits formed by chains of up to d C gates [12]. Identifying specific worst-case circuits and describing families with worst-case asymptotics remains a challenge.

5. QUANTUM SEARCH APPLICATIONS

Quantum computation is necessarily reversible, and quantum circuits generalize their reversible counterparts in the classical domain [9]. Instead of wires, information is stored on *qubits*, whose states we write as $|0\rangle$ and $|1\rangle$ instead of 0 and 1. There is an added complexity — a qubit can be in a *superposition state* that combines $|0\rangle$ and $|1\rangle$. Specifically, $|0\rangle$ and $|1\rangle$ are thought of as vectors of the *computational basis*, and the value of a qubit can be any unit vector in the space they span. The scenario is similar when considering many qubits at once: the possible configurations of the corresponding classical system are now the computational basis, and any unit vector in the linear space they span is a valid configuration of the quantum system. Just as the classical configurations of the circuit persist as basis vectors of the space of quantum configurations, so too classical reversible gates persist in the quantum context. Non-classical gates are allowed; in fact, any (invertible) norm-preserving linear operator is allowed as a quantum gate. However, quantum gate libraries often have very few non-classical gates [9]. An important example of a non-classical gate (and the only one used in this paper) is the Hadamard gate H. It operates on one qubit, and is defined as follows: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Note that because H is linear, giving the images of the computational basis elements defines it completely.

During the course of the computation, the quantum state can be anywhere in the linear space spanned by the computational basis. However, a serious limitation is imposed by quantum measurement, performed after a quantum circuit is executed. A measurement nondeterministically collapses the state onto some vector in a basis corresponding to the measurement being performed. The probabilities of outcomes depend on the measured state — basis vectors [nearly] orthogonal to the measured state are least likely to appear as outcomes of measurement. For example, if $H|0\rangle$ were measured in the computational basis, it would be seen as $|0\rangle$ half the time, and $|1\rangle$ the other half.

Despite this limitation, quantum circuits have significantly more computational power than classical circuits. In this work, we consider Grover's search algorithm, which is provably faster than any non-quantum algorithm for the same problem [4]. Grover's algorithm selects one of N unordered items that satisfy a given predicate. No structural information about the predicate is used — it is treated as a black box. Grover's algorithm completes in $\Theta(\sqrt{N})$ time, not counting the evaluation of the predicate, thereby achieving a quadratic speedup over the best possible classical algorithms if no structural information about the predicate is used in search. Grover's algorithm presupposes that the desired items are indexed from 0 to $2^n - 1$ (padding is required when N is not a power of two). Its first step is to use H gates to bring the system into a superposition of all the computational basis states. Then, a transformation called the *Grover operator* iteratively changes the state of the system so that subsequent measurement will, with high probability, yield this index. Since the result can be easily verified and since the input is classical, the procedure can be repeated until successful. If the procedure guarantees success with probability > 0.5, relatively few ($poly(n)$) repetitions are required to decrease the overall probability of failure below that of classical computers.

To implement the Grover operator, one needs an *oracle circuit* that represents the search predicate $f(x)$. This circuit transforms an arbitrary basis state $|x\rangle$ to the state $(-1)^{f(x)}|x\rangle$. The oracle is followed by (i) several Hadamard gates, (ii) a subcircuit which flips the sign on all computational basis states other than $|0\rangle$, and (iii) more Hadamard gates. A sample Grover-operator circuit for a search on 2 qubits is shown in Figure 8 and uses one qubit of temporary storage [9]. The search space here is $\{0,1,2,3\}$, and the desired indices are 0 and 3. The oracle circuit is highlighted by a dashed line. While the portion following the oracle is fixed, the oracle may vary depending on the search criterion. Unfortunately, most works on Grover's algorithm do not address the synthesis of oracle circuits and their complexity. According to Bettelli et al. [3], this is a major obstacle for automatic compilation of high-level quantum programs, and little help is available.

PROPOSITION 11. *With one temporary storage qubit, the problem of synthesizing a quantum circuit that transforms computational basis states $|x\rangle$ to $(-1)^{f(x)}|x\rangle$ can be reduced to a problem in the synthesis of classical reversible circuits [9].*

Proof: Define the permutation π_f by $\pi_f(x,y) = (x, y \oplus f(x))$, and define a unitary operator U_f by letting it permute the states of the computational basis according to π_f. The additional qubit is initialized to $|-\rangle = H|1\rangle$ so that $U_f|x,-\rangle = (-1)^{f(x)}|x,-\rangle$. If we

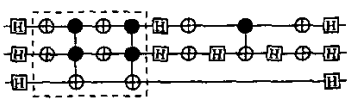


Figure 8: A Grover-operator circuit with oracle highlighted.

Size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
XOR	1	4	6	4	4	12	18	12	6	12	19	16	10	8	10	16	19	12	6	12	18	12	4	4	6	4	1
OPT T	1	4	6	4	4	12	21	24	29	33	44	46	22	5	1	0	0	0	0	0	0	0	0	0	0	0	0
OPT	1	7	21	35	36	28	28	36	35	21	7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2: Circuit size distribution of 3+2 ROM based circuits synthesized using various algorithms.

Circuit Size	0	1	2	3	4	5	6	7	Total
No. of circuits	1	7	21	35	35	24	4	1	128

Table 3: Optimal 3+1 oracle circuits for Grover's search.

now ignore the value of the last qubit, the system is in the state $(-1)^{f(x)}|x\rangle$, which is exactly the state needed for Grover's algorithm. Since a quantum operator is completely determined by its behavior on a given computational basis, any circuit implementing π_f implements U_f. In particular, since reversible gates may be implemented with quantum technology, we can synthesize U_f as a reversible logic circuit. □

Quantum computers implemented so far are severely limited by the number of simultaneously available qubits. While n qubits are necessary for Grover's search, one should try to minimize the number of additional temporary storage qubits. One such qubit is entailed by Proposition 11 to convert classical reversible circuits to alter the phase of quantum states. Another qubit is required to synthesize circuits for odd π_f, according to Proposition 4. Constructively, given π_f, we can use the algorithm of Section 4 to find an optimal circuit for it. Figure 3 gives the optimal circuit sizes of functions π_f corresponding to 3-input 1-output functions f ("3+1 oracles") which can be synthesized on four wires. These circuits are significantly smaller than many optimal circuits on four wires. This is not surprising, as they perform less computation.

In Grover oracle circuits, the main input lines preserve their input values and only the temporary storage lines can change their values. Therefore, Travaglione et al. [15] studied circuits where some lines cannot be changed even at intermediate stages of computation. In their terminology, a circuit with k lines that we are allowed to modify and an arbitrary number of read-only lines is called a k-bit ROM-based circuit. They show how to compute permutation π_f arising from a Boolean function f using a 1-bit quantum ROM-based circuit, and prove that if only classical gates are allowed, two writable bits are necessary. Two bits are sufficient if the CNT gate library is used. Their synthesis algorithms rely on XOR sum-of-products decompositions of f. We outline their method in the Appendix, in a proof of the following result.

PROPOSITION 12. *There exists a reversible 2-bit ROM based CNT-circuit computing $(x,a,b) \to (x,a,b \oplus f(x))$, where x is a k-bit input. If a function's XOR decomposition consists of only one term, let k be the number of literals appearing (without complementation) If $k > 0$ then there will be $3 \cdot 2^{k-1} - 2$ gates. [15].*

We apply the construction given in the Appendix to all 256 functions implementable in 2-bit ROM based circuits with 3 bits of ROM. The circuit size distribution is given in the line labeled XOR in Table 2. That is compared with optimal circuit sizes produced by the algorithm from Section 4. The line OPT T gives the size distribution of circuits synthesized under the restriction [15] that at most one control bit per gate be a ROM bit, which is observed by the heuristic based on XOR decomposition. This is why, for all j, the sum of the first j numbers in the OPT T line is greater than or equal to that in the XOR line. Travaglione et al [15] mention that their results do not depend on the above restriction, and the OPT line of Table 2 relaxes it.[2]

Most functions computable by a 2-bit ROM-based circuit actually require two writeable bits [15]. Whether or not a given function can be computed by a 1-bit ROM-based CNT-circuit, can be determined by the following constructive procedure. Observe that gates in 1-bit ROM circuits can be reordered arbitrarily, as no gate affects the control-bits of any other gate. Thus, whether or not a C or T gate flips the controlled bit, depends only on the circuit inputs. Furthermore, multiple copies of the same gate on the same wires cancel out, and we can assume that at most one is present in an optimal circuit. A synthesis procedure can then check which gates are present by applying the permutation on every possible input combination with zero, one, or two 1s in its binary expansion. If the value of the function is 1, the circuit must have a N, C or T gate controlled by those bits.

Observe that adding the S gate to the gate library during $k+1$ ROM synthesis will never decrease circuit sizes — no two wires can be swapped since at least one of them is a ROM wire. In the case of $k+2$ ROM synthesis, only the two non-ROM wires can be swapped, and one of them must be returned to its initial value by the end of the computation. We ran an experiment comparing circuit lengths in the 3+2 ROM-based case and found no improvement in circuit sizes upon adding the S gate, however we have been unable to prove this in the general case.

6. CONCLUSIONS

We have explored a number of promising techniques for synthesizing optimal and near-optimal reversible circuits that require little or no temporary storage. In particular, we have proven constructively that every even permutation function can be synthesized without temporary storage using the CNT gate library, and our proof is the basis of a reasonably efficient heuristic synthesis algorithm. We have also derived various equivalences among CNT-circuits that are useful for synthesis purposes. Our experimental data for optimal reversible circuits on three wires using various subsets of the CNTS library reveals some interesting characteristics of these circuits. Finally, we have applied our approach to the design of oracle circuits for a key quantum computing application, Grover's search algorithm, and obtained much smaller circuits than previous methods.

While the algorithm to synthesize optimal circuits scales better than its counterparts for irreversible computation [7], it is still limited by an exponentially growing search space. In on-going work, we are attempting to extend the proposed methods to handle larger and more general reversible circuits, with the eventual goal of synthesizing quantum circuits containing dozens of qubits.

[2]Using a circuit library with ≤ 6 gates (191Mb file, 1.5 min to generate), the OPT line takes 5 min to generate. For the OPT T line, we first find the 250 optimal circuits of size ≤ 12 (15 min) using a 6-gate library (61Mb, 5min). The remaining 6 functions were synthesized in 5 min with a 7-gate library (376Mb, 10 min).

7. REFERENCES

[1] A. Barenco et al., "Elementary Gates For Quantum Computation," *Physical Review A*, **52**, 1995, pp. 3457-3467.

[2] C. Bennett, "Logical Reversibility of Computation," *IBM J. of Research and Development*, **17**, 1973, pp. 525-532.

[3] S. Bettelli, L. Serafini and T. Calarco, "Toward an Architecture for Quantum Programming," Nov. 2001, `http://arxiv.org/abs/cs.PL/0103009`.

[4] L. K. Grover, "A Framework For Fast Quantum Mechanical Algorithms," *Proc. Symp. on Theory of Computing*, 1998.

[5] P. Kerntopf, "A Comparison of Logical Efficiency of Reversible and Conventional Gates," *IWLS 2000*, pp. 261-9.

[6] R. Korf, "Artificial Intelligence Search Algorithms", *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.

[7] E. Lawler, "An Approach to Multilevel Boolean Minimization," *JACM*, **11**, July 1964, pp. 283-295.

[8] J. P. McGregor and R. B. Lee, "Architectural Enhancements for Fast Subword Permutations with Repetitions in Cryptographic Applications," *ICCD*, 2001, pp. 453-461.

[9] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2000.

[10] M. Perkowski et al., "A General Decomposition For Reversible Logic," *Reed-Muller Workshop*, Aug. 2001.

[11] T. Sasao and K. Kinoshita, "Conservative Logic Elements and Their Universality," *IEEE Trans. on Computers*, **28**, 1979, pp. 682-685.

[12] T. Silke, "PROBLEM: register swap," December 1995, `http://www.mathematik.uni-bielefeld.de/~silke/PROBLEMS/bit_swap`

[13] Z. Shi and R. Lee, "Bit Permutation Instructions for Accelerating Software Cryptography," *IEEE Intl. Conf. on Application-specific Systems, Architectures, and Processors*, July 2000, pp. 138-148.

[14] T. Toffoli, "Reversible Computing," *Tech. Memo MIT/LCS/TM-151*, MIT Lab for Comp. Sci, 1980.

[15] B. Travaglione et al. "ROM-based computation: Quantum Versus Classical," 2001. `http://arxiv.org/abs/quant-ph/0109016`

[16] S. Younis and T. Knight, "Asymptotically Zero Energy Split-Level Charge Recovery Logic," *Workshop on Low Power Design*, 1994.

Appendix

Below we state and prove technical results used in Section 3 and then detail a proof of Propostion 12.

PROPOSITION 13. *For $n \geq 5$, we can write any permutation in A_n as the product of no more than n pairs of disjoint transpositions.*

Proof: Fix $\pi \in A_n$. Then take the cycle decomposition of π and decompose each cycle into transpositions to write π as a product of $c(\pi) \leq n$ transpositions. Since π is even, we know $c(\pi) = 2k$ for some k. Pair up the $2i$-th and $(2i+1)$-st transpositions. Some of these pairs can not be disjoint, but since $n \geq 5$ we can write $(a,b)(a,c) = (a,b)(d,e)(d,e)(a,c)$ where $d \neq e$ are distinct from a,b,c. Thus breaking up non-disjoint pairs, we write π as a product of $2k = c(\pi) \leq n$ pairs of transpositions. □

PROPOSITION 14. *Let $n \geq 4$, and a,b,c,d be distinct integers between 0 and $n-1$. Then there exists a constructable permutation $\pi \in A_{2^n}$ such that $\pi(a) = 0$, $\pi(b) = 1$, $\pi(c) = 2$, and $\pi(d) = 3$. It takes at most $2n$ N gates, $4(n+1)$ C gates, and $2(n-2)$ T gates.*

Proof: Start with an empty circuit and place N gates on every line corresponding to a 1 in the binary expansion of a. Let π_0 be the permutation performed by the circuit so far; $\pi_0(a) = 0$. Since $b \neq a$, so $\pi_0(b) \neq 0$ and therefore $\pi_0(b)$ has at least one 1 in its binary expansion. Say it's on the k-th line; then using C gates controlled on the k-th line, flip any other non-zero bits of b'. Finally, if $k \neq 1$, swap the k-th bit and the 0-th bit. This can always be done using 3 C gates, as in Figure 2. In this case, since we know that the bottom bit is 0 and the k-th bit is 1, we need only 2.

Let π_1 be the permutation performed by the circuit so far. by construction, $\pi_1(b) = 1$, and since C gates fix 0, we have $\pi_1(a) = \pi_0(a) = 0$. As before, $c \neq b, a \implies \pi_1(c) \neq 1,0$ hence $\pi_1(c)$ has a 1 somewhere in its binary expansion other than the lowest bit, say in the p-th bit. Using the algorithm of the previous paragraph, flip every other bit to 0 and then swap the p-th and 2-nd bit; we note that again we have not affected 0, and none of our C gates have been controlled on the bottom line, we cannot move 1. The permutation π_2 performed by the circuit thus far has the property that $\pi_2(c) = 2$, $\pi_2(b) = 1$, $\pi_2(a) = 0$.

Finally, observe that $\pi_2(d) \geq 3$; if it is in fact 3 then we are done, if not then we have $\pi_2(d) \geq 4$, and some bit in the binary expansion of $\pi_2(d)$ other than the lowest two bits must be 1; let it be the q-th bit. Then using C gates controlled the q-th bit, flip the bottom two wires to 1 if necessary, and use T gates controlled on these bottom two bits to clear off the rest of the wires. We are now done, as none of these gates affect $0,1,2$, and this subcircuit sends $\pi_2(d) \to 3$. A careful count of the gates used verifies the final claim of the proposition. □

PROPOSITION 12. *There exists a reversible 2-bit ROM based CNT-circuit computing $(x,a,b) \to (x,a,b \oplus f(x))$, where x is a k bit input. If a function's XOR decomposition consists of only one term, let k be the number of literals appearing (without complementation) If $k > 0$ then there will be $3 \cdot 2^{k-1} - 2$ gates. [15].*

Proof: Assume we are given an XOR sum-of-products decomposition of f. Then it suffices to know how to transform $(x,a,b) \to (x,a,b \oplus p)$ for an arbitrary product of uncomplemented literals p, because then we can add the terms in an XOR decomposition term by term. So, without the loss of generality, let $p = x_1 \ldots x_m$. Denote by $T(a,b;c)$ a T gate with controls on a,b and inverter on c. Similarly, denote by $C(a;b)$ a C gate with control on a and inverter on b. Number the ROM wires $1 \ldots k$, and the non-ROM wires $k+1$ and $k+2$. Let us first suppose that there is at least one uncomplemented literal, and put a $C(1,k+2)$ on the circuit; note that $C(1,k+2)$ applied to the input (x,a,b) gives $(x,a,b \oplus x_1)$. We will write this as $C(1,k+2) : (x,a,b) \to (x,a,b \oplus x_1)$, and denote this operation by V_1. Then, we define the circuit V_2' as the sequence of gates $T(2,k+2,k+1)V_0T(2,k+2,k+1)V_0$, and one can check that $V_2' : (x,a,b) \to (x,a \oplus x_1x_2,b)$. We define V_2 by exchanging the wires $k+1$ and $k+2$; clearly $V_2 : (x,a,b) \to (x,a,b \oplus x_1x_2)$. In general, given a circuit $V_l : (x,a,b \oplus x_1 \ldots x_{l-1}) \to (x,a \oplus x_1 \ldots x_l)$, we define $V_{l+1}' := T(l+1,k+2,k+1)V_lT(l+1,k+2,k+1)V_l$; one can check that $V_{l+1}' : (x,a,b) \to (x,a \oplus x_1 \ldots x_{l+1},b)$. Define V_{l+1} by exchanging the wires $k+1$ and $k+2$; then clearly $V_{l+1} : (x,a,b) \to (x,a,b \oplus x_1 \ldots x_{l+1})$. By induction, we can get as many uncomplemented literals in this product as we like. □