Synthesis of Quantum Multiple-Valued Circuits

D. Michael Miller, Dmitri Maslov

Department of Computer Science

University of Victoria

Victoria, BC, V8W 3P6 Canada

Gerhard W. Dueck

Faculty of Computer Science

University of New Brunswick

Fredericton, NB, E3B 5A3 Canada

February 16, 2004

Abstract

An r-valued m-variable reversible logic function maps each of the r^m input patters to a unique output pattern. The synthesis problem is to realize a reversible function by a cascade of primitive reversible gates.

In this paper, we present a simple heuristic algorithm that exploits the bidirectional synthesis possibility inherent in the reversibility of the specification. The primitive reversible gates considered here are MVL extensions of the well-known binary Toffoli gates. We analyze the structure of the gates that we use and show how these gates can be easily simulated in a quantum technology.

We present exhaustive results for the 9! 2-variable 3-valued reversible functions comparing the results of our algorithm to optimal results found by breadth-first search. We also show results for specific ternary examples including showing how the presented technique can be applied to the synthesis of a 3-input, 3-valued adder which is not itself a reversible problem. When the circuit for the full adder is synthesized, we use it as an example of an approach to a circuit simplification, called the templates tool. Formalization and proper investigation of the templates simplification tool proposed in this paper is still under investigation.

Keywords: logic synthesis, quantum circuits, MVL, reversible logic.

1 Introduction

A binary or MVL circuit is reversible if it maps each input pattern to a unique output pattern. Landauer [9] proved that traditional binary irreversible gates lead to power dissipation in a circuit regardless of its implementation technology. Recently, Zhirnov *et al.* [19] calculated that power dissipation in future CMOS (scaled for the year 2016 in accordance with the year 2001 International Technology Roadmap for Semiconductors) leads to impossible heat removal, and thus the impossibility of further speeding up CMOS technology devices. Bennett [4] showed that for power not to be dissipated it is necessary that a binary circuit be build from reversible gates. Binary reversible circuits have been studied for their potential application in quantum computation [14], low-power CMOS design and optical computing.

Here[1] we consider the synthesis of a reversible MVL circuit as a composition of reversible MVL logic gates. We make the following assumptions that are dictated by one of the probable target technologies (quantum), and, most likely, some other reversible technologies:

- fan-out between gates is not permitted;
- loops are not permitted; and
- permutation of connections between gates is permitted.

We employ MVL reversible gates that are extensions of the binary reversible NOT, Feynman [5] and Toffoli gates [17]. The gates considered are those introduced by De Vos et al. [18] and simple extensions of those gates. We describe the gates that we use constructively, that is, assuming that the quantum cost can be efficiently approximated by the number of one-qubit and two-qubit controlled-V quantum transformations [3, 14]. We show how to simulate our MVL gates with the above operations.

[1] A preliminary version of this work [13] is to be presented at ISMVL-04.

The gates considered here are not the only possible MVL reversible gates nor are they the only possible extension to the Toffoli gate. Picton [15, 16] introduced an MVL generalization of the binary Fredkin gate. Al-Rabadi [2] has considered a generalization of the Toffoli gate where EXOR is replaced by mod-sum. Khan *et al.* [8] have considered several MVL (ternary) reversible gates. The set of gates we use here is of interest because of its relative simplicity and consistency and because preliminary investigations indicate that implementation should be relatively efficient. The reader will observe that due to the simple nature of our synthesis method it can be extended to other sets of primitive reversible gates.

Synthesis of reversible logic is very different from conventional synthesis. Since loops are not permitted, a reversible logic circuit can be specified as a simple sequence of gates. Further, since fan-out is not permitted, and assuming an appropriate technology, a reversible logic circuit can realize the inverse specification simply by applying the gates in the reverse order since the gates themselves implement reversible functions. Hence, synthesis can be carried out from the inputs toward the outputs or from the outputs toward the inputs. The method presented here synthesizes the circuit by working in both directions simultaneously. In addition, it is advantageous to synthesize a circuit for a given specification and also for its inverse taking the solution to be the simpler result. The synthesis method presented here is based on the binary method developed by the authors in [12], but there are certain novel issues to deal with in the MVL case.

The background on reversible logic necessary for this paper is outlined in Section 2. The MVL reversible gates used in this work are described in Section 3. In Section 4 we analyze how the ternary computations and the gates that we propose to use for the synthesis can be implemented in a quantum technology. The interested reader is referred to [14] for extensive background. Section 5 presents our synthesis algorithm. An example illustrating the operation of the algorithm is worked in some detail in Section 6. Experimental results are given in Section 7 and an irreversible example, the 3-valued full adder is shown. In Section 8 we further explore the example from Section 7 and show how the synthesized circuit can be simplified. The paper concludes with observations and ideas for further research in Section 9.

2 Background

Definition 1. An m-input, m-output, (written $m \times m$) totally-specified MVL function is reversible if it maps each input assignment to a unique output assignment. ►

A reversible function thus defines a permutation of the input patterns and there are clearly $r^m!$ r-valued, $m \times m$ reversible functions.

Definition 2. An n-input, n-output gate is reversible if it realizes a reversible function. ►

The synthesis problem is how to realize a given reversible specification using a basic set of reversible gates.

A variety of binary reversible gates have been considered. The common NOT gate realizes a reversible function. Another binary reversible gate is the Fredkin gate [6] which has three inputs and three outputs. The first input is passed through unaltered. The second and third pass through unaltered if the first input is 0 and are exchanged if the first input is 1. There is also the family of Toffoli gates [17] defined as follows:

Definition 3. An $n \times n$ Toffoli gate passes the first $(n-1)$ (control) lines through unchanged, and inverts the n^{th} line (target) if the control lines are all 1. ►

NOT, Fredkin and Toffoli gates are all self-inverse since applying two gates of the same type and size in order results in an identity permutation.

3 MVL Reversible Gates

A. De Vos *et al.* [18] have considered the cycle and negation operations in Table 1, denoted $C1$ and N respectively, and the controlled versions of those gates, denoted $CC1$ and CN, given in Table 2, as generators of the group of all 2×2 3-valued reversible logic functions. The notation

used here is that symbols such as x and y denote the line values on one side of a reversible gate while x^+ and y^+ denote the corresponding values on the other side of the gate.

We have written a program to determine the number of gates for circuits realizing the $9! = 362,880$ 2×2 3-valued reversible logic functions for different sets of basic reversible gates. The program performs a breadth-first search so the first circuit found for each function uses minimal gates. Results are shown in Table 3.

	C1	**N**
x	x^+	x^+
0	1	2
1	2	1
2	0	0

Table 1: MVL gates $C1$ and N.

		CC1		**CN**	
x	y	x^+	y^+	x^+	y^+
0	0	0	0	0	0
0	1	0	1	0	1
0	2	0	2	0	2
1	0	1	1	1	2
1	1	1	2	1	1
1	2	1	0	1	0
2	0	2	0	2	0
2	1	2	1	2	1
2	2	2	2	2	2

Table 2: MVL gates $CC1$ and CN.

Cyclic inversion ($C1$) together with controlled negation (CN) is the only operation pair that can realize all 362,880 functions. As shown, the circuits can be quite long. Adding, controlled cycle ($CC1$) and negation (N) in turn both improve the results. The latter is the set of operators suggested by De Vos et al. [18].

The rightmost column in Table 3 shows the further improvement gained by adding cycle by 2 ($C2$) and controlled cycle by 2 ($CC2$) defined in Tables 4 and 5 respectively. Clearly, N and CN are self-inverse. $C1$ and $C2$ are the inverse of each other. Hence, $CC1$ and $CC2$ are the inverse of each other. In the next section we also show that the gates $C2$ and $CC2$ are no harder to simulate in quantum technology than the gates $C1$ and $CC1$ correspondingly. Thus, the usage of gates $C2$ and $CC2$ is preferred to the usage of a reversible circuit from the elements $C1$, N, $CC1$ and CN. Negation can be extended to any r-valued logic as $\bar{x} = (r-1)-x$. There are $(r-1)$ cyclic inversions for r-valued logic defined in the obvious way. Controlled cycles and controlled negation can be generalized to the $n \times n$ cases for r-valued logic as specified in the following definition.

Definition 4. An $n \times n$ r-valued controlled unary gate passes the first $(n-1)$ (control) lines through unchanged, and applies a specified unary operation to the n^{th} line if the control lines are all 1, otherwise the target line is passed through unaltered. ►

The Fredkin gate can clearly be extended to the MVL case (and it also turns out that in a quantum realization the MVL Fredkin gate is the same as the regular 2-CNOT 1-Toffoli binary Fredkin gate), but we do not consider that in this paper.

4 Quantum Implementation Analysis

A natural and important question would be if reversible ternary computations are supported by an existing technology. Since reversible logic is often considered in conjunction with quantum computations, we illustrate how quantum technology allows inexpensive modeling of the reversible ternary computations. For that, we need to:

- show how to code ternary constants 0, 1 and 2 with quantum bit states;
- show how to measure the output once the coding is provided;
- build the quantum gates $C1$, $C2$, N, D and E (the last two are defined in Table 6) which would act like the corresponding ternary gates when their domain is restricted to the set of quantum codings of ternary 0, 1 and 2;
- estimate the quantum implementation costs of these gates and their controlled versions.

We briefly review the basic concepts of quantum computation. For a more detailed and formal introduction we refer reader to [14]. A single quantum bit (qubit) has one of the two possible states, 0 or 1, traditionally depicted as $|0\rangle$ and $|1\rangle$ correspondingly. Due to the physical nature of the object that represents a quantum bit, its measurement gives the result of $|0\rangle$ or $|1\rangle$ with some probabilities p and q ($p + q = 1$). Further, it turns out that the state of a single qubit is a linear combination $\alpha|0\rangle + \beta|1\rangle$ (also written as a vector (α, β) in the basis $\{|0\rangle,\ |1\rangle\}$), where α and β are complex numbers called amplitudes, and $|\alpha|^2 + |\beta|^2 = 1$. Real numbers $|\alpha|^2$ and $|\beta|^2$ represent the probabilities of reading the states $|0\rangle$ and $|1\rangle$ upon measurement. The state of a quantum system with $n > 1$ qubits is described as an element of tensor product of the single state spaces, thus, being a structure that can be mapped into a normalized vector of length 2^n, called the state vector. The quantum system evolution allows changes of the state vector given by multiplications by $2^n \times 2^n$ unitary matrices. This principle states the conditions upon which a transformation can be performed, but does not tell how hard it is to do the transformation. Further, for different quantum technologies the same gate may have very different costs associated with how hard it is to implement it.

Given that, for the general case it makes sense to suppose that one qubit and two qubit controlled-V operations (the state of a single qubit is changed according to the form of 2×2 unitary matrix V iff the state of the controlling qubit is $|1\rangle$) have a low cost and estimate the cost of a quantum computation by the number of one qubit and controlled-V operations needed for the computation. This will give a very crude approximation, but may be a good starting point.

Measurement of a one qubit system $\alpha|0\rangle + \beta|1\rangle$ can be understood as a projection, that is, reading $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Once the measurement is done, and the result of $|0\rangle$ or $|1\rangle$ read, every next measurement gives the same result as the one read before. Thus, the meaningful measurement of a single qubit can be done only once. However, it is possible to make the quantum computation on several quantum systems in parallel and then do the measurements. Then, the average of these measurements gives an estimation of values $|\alpha|^2$ and $|\beta|^2$. Projections onto non-basis vectors are also possible.

There are many approaches to the physical realization of quantum computations. Among them are (liquid) NMR, optical, trapped ion, neutral atom, solid state, superconducting and other technologies [7]. All are so different, that it is difficult to make general assumptions and to develop general methods that will be suitable for all technologies. In this paper, we consider NMR technology as a primary application. Among the named technologies NMR technology appears to be one of the most promising technologies, also one that has been most investigated. Major problem includes absence of an efficient initial state preparation procedure, during which the signal loss would not be tremendously high (exponential in number of qubits as $n2^{-n}$ for the current inefficient preparation procedures [14]). This prevents construction of large NMR quantum computers at the present time. However, we hope to partially overcome this obstacle by considering MVL computations that require less qubits in comparison to the binary case in the following sense. If, for example, the domain size for a problem is required to contain 1000 instances, at least 10 ($= \lceil \log_2(1000) \rceil$) qubits should be used in binary case, whereas in ternary this number is only 7 ($= \lceil \log_3(1000) \rceil$). Currently, the practical number of qubits in NMR technology is 7 [1].

We suggest that the ternary values 0, 1 and 2 are coded by the states $\frac{\sqrt{3}|0\rangle - |1\rangle}{2}$, $|1\rangle$ and $\frac{-\sqrt{3}|0\rangle - |1\rangle}{2}$ of a single qubit correspondingly (Figure 1). Further, we refer to ternary 0, 1 and 2 as the corresponding quantum states. The next two paragraphs explain why we chose the above encoding.

In order to be able to read and reliably distinguish the ternary states in NMR, when the computation stage is finished, divide the set of all molecules into two parts. Then, in the first part we

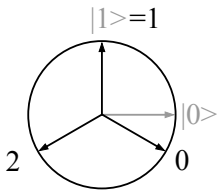

Figure 1: Ternary states.

measure ternary 1 and on the second measure ternary 2. For simplicity consider the first part, where ternary value 1 is measured. If the actual state of the qubit was 0, the measurement will produce reading of the signal with intensivity $\left(-\frac{1}{2}\right)^2 = \frac{1}{4}$. If the actual value was 2, the reading will be the same, $\frac{1}{4}$. Only if the actual value was 1, the measurement signal will be 1. Thus, the first part distinguishes value 1. Note, that from the point of view of such measurement the states 0 and 2 will be indistinguishable. Similarly, it can be concluded that the second part distinguishes value 2. It is easy to see that if none of the two parts distinguished their value (that is, produced a unit signal upon measurement), the actual value of the measured qubit was 0.

In the following we show how the ternary gates $C1$, $C2$, N, D and E can be simulated by circuits with simple quantum gates. $C1$ gate is quantum rotation gate $R(2\pi/3)$, $C2$ is rotation $R(-2\pi/3)$, N is $-Z$, where Z is a Pauli-Z gate. These are very simple gates and the literature [14, 3] suggests a low cost for them and uses them as elementary building blocks. We assign the cost of one to each of these three operations. Gates D and E are somewhat more expensive. But, they can be simulated using gates $C1$, $C2$ and N. Gate D can be simulated by the network N $C1$ and E by the network N $C2$. Thus, their costs are two each. The literature [14] has no explanation on how to construct controlled Pauli-Z, but uses such gate. Similarly, we assume that the cost of each gate $CC1$, $CC2$ and CN is one. Then, the gates controlled-D and controlled-E can be simulated with the circuits CN $CC1$ and CN $CC2$ respectively, giving the cost of 2 for these implementations. Gates $C1$, $C2$ and N with two controls each can be simulated by the circuit with 5 gates, illustrated on page 182 of [14]. Using the simulation of one bit gates D and E by the gates $C1$, $C2$ and N, one can easily build circuits with cost 8 for the gates D and E with two controls. The gates with larger

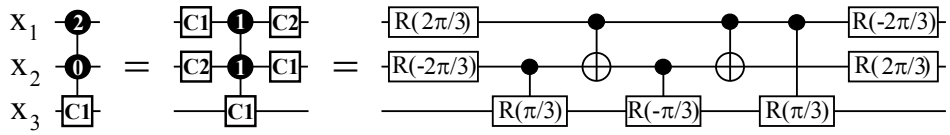

Figure 2: The structure of the $C1(x_1 = 2,\ x_2 = 0;\ x_3)$ gate.

sets of controls can be simulated by applying the technique of [3]. For the later considerations, it is important to notice that the gates $C1,\ C2$ and N with 3 unit controls can be simulated with 13 operations.

In addition to the gates analyzed above, the synthesis algorithm presented in Section 5 uses gates with non-unit controls. Such gates can be simulated by applying the rotations $C1$ and $C2$ before and after the controlling bits are used. We illustrate the above simulations by constructing the $C1(x_1 = 2,\ x_2 = 0;\ x_3)$ gate, a gate that applies $C1$ transform to the variable x_3 iff variable x_1 carries value 2 and variable x_2 carries value 0, in Figure 2. The first part of Figure 2 shows how we depict the gate, second part shows how to simulate the gate when only unit controls are available, the third part shows the quantum structure of the gate. It can be observed that $C1(x_1 = 2,\ x_2 = 0;\ x_3)$ can be simulated by the circuit with 9 simple quantum gates.

5 A Synthesis Method

An $m \times m$ reversible MVL specification is a mapping

$$F : Q \rightarrow Q$$

where Q is the set of r^m m-tuples of r values. We only consider the totally-specified case in this paper. As F is reversible, there is of course a corresponding inverse mapping

$$F^{-1} : Q \rightarrow Q$$

We will write $F(x) = y$ (conversely $F^{-1}(y) = x$) where $x, y \in Q$ and shall denote the elements of Q as the ordered set $\{q_0,\ q_1, \ldots\ q_{r^m-1}\}$ hence q_i is the m-ary r-valued expansion of integer i.

Our synthesis method operates by finding a sequence of MVL reversible gates whose effect is to transform F (and of course F^{-1}) to the identity mapping. It is based on the binary synthesis approach we developed in [12].

There are two aspects of our approach which must be noted at this point. First, our algorithm allows control values to be any non-zero value. The reason will become obvious in the description of the algorithm. However, gates with the zero-valued controls are useful and they will be considered in Section 8. This extension leads to the following:

Definition 5. A generalized $n \times n$ r-valued controlled unary gate passes the first $(n-1)$ lines through unchanged, and applies a specified unary operation to the n^{th} line if each of the control lines equals the control value c $(0 \leq c \leq r-1)$ specified for that line; otherwise the target line is passed through unaltered. ▶

Also, because our algorithm processes the specification in a specified order, we need an additional operation D (see Table 6) which we also use as a controlled gate.

Definition 6. For a reversible specification F, the distance between F and the identity is given by

$$\Delta(F) = \sum_{j=0}^{r^m-1} d(q_j, F(q_j)),$$

where for two r-valued m-tuples a and b

$$d(a,b) = \sum_{k=0}^{m-1} |a_k - b_k|.$$

▶

Synthesis Procedure

Input: reversible specification F_0.

Output: an ordered set of gates G_{r^m} that implements the initial reversible specification.

1. $i = 0$ and $G_0 = \emptyset$;

2. if $F_i(q_i) = q_i$ skip to step 9;

3. let S be an ordered set of gates to map $F_i(q_i)$ to q_i;

4. let T be an ordered set of gates to map $F_i^{-1}(q_i)$ to q_i;

5. for $j = 0$ to i set $F_{i+1}(q_j) = q_j$;

6. if $|S| < |T|$

   (a) for $j = i + 1$ to $r^m - 1$ set $F_{i+1}(q_j)$ the result of applying the gates in S (in order) to $F_i(q_j)$;

   (b) set G_{i+1} to $G_i| < \ldots, S_1, S_0 >$ (note, | denotes concatenation);

7. if $|S| > |T|$

   (a) for $j = i + 1$ to $r^m - 1$ set $F_{i+1}^{-1}(q_j)$ the result of applying the gates in T (in order) to $F_i^{-1}(q_j)$;

   (b) set G_{i+1} to $< T_0, T_1, \ldots > |G_i$;

8. if $|S| = |T|$ we apply either 6(a) and (b) or 7(a) and (b) whichever yields the smallest $\Delta(F_i)$ (in the event of a tie, 6(a) and (b) are applied);

9. $i = i + 1$;

10. if $i < r^m - 1$ go to step 2.

The following notes clarify the steps of the algorithm.

1. Initialization step.

2. If $F(q_i) = q_i$ no transformation is required for this pass.

3. The gates to map $F_i(q_i)$ to q_i are chosen to (a) minimize the number of gates and to (b) ensure that when they are applied they have no affect for any $j < i$. The actual procedure for choosing the gates is given in detail below.

4. This is analogous to step 3 but is being applied in the opposite direction.

5. Due to the gate selection process F_{i+1} is identical to F_i for q_0 to q_i.

6., 7. and 8. S is a set of gates that map an output pattern to match the corresponding input pattern. T is a set of gates that map that input pattern to match the output pattern. The algorithm selects the smaller set and in the case of a tie the set that results in a specification with smallest distance to the identity mapping. Note that gates mapping an output pattern to match the input pattern are appended to the end of G and in reverse order (step 6b). Conversely, gates mapping an input pattern to match the output pattern are appended to the beginning of G in order.

9. and 10. are iteration control.

The key to the synthesis algorithm is the selection of a set of gates to map an m-tuple q_k to another q_i. By construction, $k > i$. Also, the gates must be chosen so that they have no affect on any $q_j,\ j < i$. Both of these stipulations arise from the fact the algorithm goes through the specification in order. The gate selection is illustrated below in terms of a generic reversible specification F which of course can be the inverse of another specification.

Gate Selection Subprocedure

Input: a reversible specification F and an i for which we want to select gates to transform $F(q_i)$ to q_i.

Output: an ordered set of gates G performing the required transformation.

1. For ease of notation let $a = F(q_i)$ and let $b = q_i$. Set $G = \emptyset$.
2. Let $k = 0$.
3. If $a_k = b_k$ skip to step 3.
4. Select a gate g that transforms a_k to b_k with control values (if needed) being the smallest number of $a_p,\ p \neq k$ so that the value of the control set is greater or equal b. The gate chosen in the event of a choice is the one that yields a transformed specification such that $\Delta(F^t)$ is minimal.

5. Set G to $G|g$.

6. Assign $k = k + 1$.

7. If $k < m$ go to step 3.

Step 4 is the core of the above procedure. A difference between the binary and the MVL situations is the variety of choice for mapping one value to another. For example, the case for ternary ($r = 3$) is shown in Table 7. In the event there is a choice in constructing the control set, we choose higher control values.

It is very important to note that in Table 7 we do not use $C1$ for $1 \rightarrow 2$ transformation or $C2$ for $2 \rightarrow 1$ transformation as these would always modify an entry earlier in the specification. This is an artifact of our simple algorithm processing the specification in order. We include E in Table 7 for completeness but note that it is not used in our current implementation.

We can further exploit reversibility by applying the algorithm to F and then to F^{-1}. Even though the algorithm itself exploits reversibility, it is heuristic and applying it to both the original and the inverse can produce a different circuit. The circuit for F^{-1} can of course simply be applied in reverse to realize F.

Both the synthesis and gate selection procedures are greedy in that they make choices to optimize the circuit based on only local information. No backtracking or lookahead is used.

6 example

A trace of a 2-variable, 3-valued example will illustrate the operation of the synthesis algorithm. The initial specification is given in Table 8(a). The choice is to map output pattern 21 to 00 or input pattern 21 to 00. While these seem equivalent, they are not. The synthesis procedure chooses to map the input pattern 21 as this yields the resulting specification with smaller $\Delta(F)$. The gates required are $N(x)$ and $C2(y)$. The resulting specification is given in Table 8(b) with bold type denoting the changes which in this case are to the input. The entries 00, 01, ... , 20

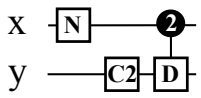

Figure 3: Resulting circuit.

are properly aligned. The next choice is to map 21 to 22 on either the input or the output side. Both require a single gate which is a controlled D applied to y with control $x = 2$. The resulting specification is given in Table 8(c) and is the identity so the process is complete. The circuit is shown in Figure 3.

7 Experimental Results

We have implemented the synthesis method described in Section 4 in C. Even on a 750MHz PC with 256MB RAM running Windows XP, the computation time for functions of a few variables is negligible. The exhaustive enumeration described below takes a few minutes. We have applied our synthesis algorithm to the $9! = 362,880$ 2×2 3-valued reversible logic functions. The results are shown in Table 9. As noted above the algorithm allows control values of 1 and 2 and uses D operations. E operations are not used. The table shows the results for applying the algorithm to F only, and the results for applying the algorithm to F and F^{-1} and taking the smaller circuit. For comparison, we include the optimal results found by breadth-first search when control values of 1 and 2 and D operations are permitted.

We present the case of a 3-valued full adder as an example of applying the above reversible logic synthesis technique to an irreversible specification. The full adder has three inputs x_0, x_1, x_2 and two outputs which are the *sum* and *carry* defined in the obvious way (Table 10).

To apply the above methods to this specification we must embed it in a larger reversible specification. Looking at the truth table for the full adder we see that output pattern 10 appears 7 times. All the output patterns in the reversible specification must be unique. This requires that we add at least two "garbage" outputs. The term garbage refers to the fact those outputs are not part of

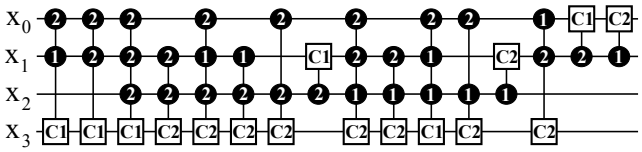

Figure 4: Circuit for the 3-valued 3-bit full adder.

the useful output set.

The reversible specification must have the same number of inputs and outputs, so we must add at least one "constant" input x_3 which we add as the most significant entry in the truth table and which we define so that setting it to 0 yields the correct *sum* and *carry* functions.

The difficult part is how to define the two garbage outputs so that the resulting specification is reversible and to minimize the resulting circuit. We draw upon our experience of the full adder in the binary case [12]. The reversible specification is given by

$$\begin{aligned} x_0^+ &= sum(x_0, x_1, x_2) \\ x_1^+ &= x_1 \oplus x_2 \\ x_2^+ &= x_2 \\ x_3^+ &= carry(x_0, x_1, x_2) \oplus x_3 \end{aligned}$$

where $\oplus$ denotes modulo-3 sum. Note that the ordering of the outputs is important and that the two garbage outputs are placed between the *sum* and *carry*. The circuit only produces the correct carry when $x_3 = 0$. Choosing the best output ordering is in general difficult. Here it was done by inspection. Our synthesis algorithm is sufficiently fast that for small problems, one could consider all possible output permutations. How to choose an output order for a large problem is an open question.

Applying the synthesis method to the specification described above yields the circuit shown in Figure 4. 16 gates are required. The algorithm gives 27 gates for the inverse specification.

8 Future Work: Templates Simplification Tool

Once the algorithm has terminated with a valid circuit, the latter can be simplified by matching certain pieces of this network and replacing them with equivalent but smaller pieces. Application of such procedure may result in a smaller circuit. We systemize this idea by introducing a template simplification tool [11].

A **template of size** m is a cascade of gates $G_0\ G_1 ... G_{m-1}$ which realizes the identity function. Any template of size m should be independent of smaller size templates, *i.e.* application of smaller size templates does not decrease the number of gates in a size m template. Given a template $G_0\ G_1 ... G_{m-1}$, its application for a parameter k, $m/2 \leq k \leq m$ and any i such that $0 \leq i \leq m$ is:

$$G_i\ G_{i+1\ (\mathrm{mod}\ m)} ...\ G_{i+k-1\ (\mathrm{mod}\ m)} \rightarrow$$

$$G^{-1}_{i-1\ (\mathrm{mod}\ m)}\ G^{-1}_{i-2\ (\mathrm{mod}\ m)} ...\ G^{-1}_{i+k\ (\mathrm{mod}\ m)}.$$

Application of such a template is replacement of a piece of network with another piece. If the piece to be replaced is larger or costly, then template application leads to the network simplification. However, before we proceed, correctness of template application has to be shown, that is equity of the first and second part of the transformation. The following set of equalities proves the correctness for the case where $i + k - 1 \geq m$. For the case $i + k - 1 < m$ the proof is similar.

$$G_0\ G_1 ...\ G_{m-1} = Id$$

$$G_0\ G_1 ...\ G_{m-1}\ (G_0\ G_1\ ...G_{i+k-1\ (\mathrm{mod}\ m)}) =$$

$$G_0\ G_1\ ...G_{i+k-1\ (\mathrm{mod}\ m)},$$

$$G_0\ G_1 ...\ G_{i-1\ (\mathrm{mod}\ m)}\ G_i\ G_{i+1\ (\mathrm{mod}\ m)}\ ...G_{i+k-1\ (\mathrm{mod}\ m)} =$$

$$G_0\ G_1\ ...G_{i+k-1\ (\mathrm{mod}\ m)},$$

$$(G^{-1}_{i-1\ (\mathrm{mod}\ m)}\ ...G^{-1}_1\ G^{-1}_0)\ G_0\ G_1 ...\ G_{i-1\ (\mathrm{mod}\ m)}\ G_i$$

$$G_{i+1\ (\mathrm{mod}\ m)}\ ...G_{i+k-1\ (\mathrm{mod}\ m)} = (G^{-1}_{i-1\ (\mathrm{mod}\ m)}\ ...G^{-1}_1\ G^{-1}_0)$$

$$G_0\ G_1\ ...G_{i+k-1\ (\mathrm{mod}\ m)},$$

$$G_i\ G_{i+1\ (\mathrm{mod}\ m)}... G_{i+k-1\ (\mathrm{mod}\ m)} =$$

$$G^{-1}_{i-1\ (\mathrm{mod}\ m)}\ G^{-1}_{i-2\ (\mathrm{mod}\ m)}... G^{-1}_{i+k\ (\mathrm{mod}\ m)}.$$

If the set of such templates is constructed, it can be used to simplify circuits. We propose a circuit simplification using templates tool that consists of the two parts. The first uses templates that consist of the controlled variations of the reversible ternary gates $C1$, $C2$, N, D and E, called the reversible template application. The second uses templates that consist of the quantum one-qubit and two-qubit controlled-V gates. This second part (as proposed in [10]) is called the quantum template simplification. Before the quantum templates are applied each reversible ternary gate is substituted with its equivalent quantum representation, discussed in Section 4 of this work. Our future work includes construction of the classes of templates and their application to the circuit simplification. In this paper we only illustrate what simplification can be done using only a few reversible templates and none of the quantum templates.

Example 1. Take the circuit for the 3-bit full adder in the Figure 4. It can be easily calculated that the number of quantum operations (as discussed in Section 4) for this circuit is $(5 + 1 * 2) + (5 + 2 * 2) + (13 + 3 * 2) + (5 + 2 * 2) + (13 + 2 * 2) + (5 + 1 * 2) + (5 + 2 * 2) + (1 + 1 * 2) + (13 + 2 * 2) + (5 + 1 * 2) + (13 + 1 * 2) + (5 + 1 * 2) + (1) + (5 + 1 * 2) + (1 + 1 * 2) + (1) = 138$.
Consider the following 3 template classes.

1. $Ck(S; v)\ Cl(S; v)$, where k and l are the numbers such that $\{k,\ l\} = \{1,\ 2\}$, S is any set of controls and v is the target variable of the gate. This class lists some sequences of the two gates that can be deleted from the network without changing its output.

2. $Ck(S; v)\ Ck(S; v)\ Ck(S; v)$, where k is a number from the set $\{1,\ 2\}$, S is a set of controls and v is a variable. Essentially, it says that application of two identical $C1$ gates in a row is equivalent to the application of one $C2$ gate, and symmetrically, application of two equal $C2$ gates is equivalent to the application of one $C1$ type gate.

3. $Ck(S, v_1 = x; v_2)\ Ck(S, v_1 = y; v_2)\ Ck(S, v_1 = z; v_2)\ Cl(S; v_2)$, where k and l are the numbers such that $\{k,\ l\} = \{1,\ 2\}$, $x,\ y$ and z are the different ternary numbers, S is any set of controls, and v_1 and v_2 are any two different variables.

4. $Ck(S_1; v_1)\ Cl(S_2; v_2)\ Ck(S_1; v_1)\ Cl(S_2; v_2)$, where k and l are the numbers from the set $\{1,\ 2\}$, S_1 and S_2 are the sets of controls, v_1 and v_2 are the variables (not necessarily different), v_1 is not listed in the set S_2 and v_2 is not listed in the set S_1. This class defines some of the cases when the gates in a circuit can be moved one past the other. We call it the moving rule.

The third class can be applied to the first and the second gates of the circuit in Figure 5A when the parameters are chosen so that $k = 1,\ S = \{x_0 = 2\},\ v_1 = x_1,\ v_2 = x_3,\ x = 1$ and $y = 2$. Then, its application transforms the sequence of gates $C1(x_0 = 2,\ x_1 = 1;\ x_3)\ C1(x_0 = 2,\ x_1 = 2;\ x_3)$ to the sequence $C2^{-1}(x_0 = 2;\ x_3)\ C1^{-1}(x_0 = 2,\ x_1 = 0;\ x_3) = C1(x_0 = 2;\ x_3)\ C2(x_0 = 2,\ x_1 = 0;\ x_3)$. This transformation does not decrease the number of gates, but decreases the number of controls in the gates, resulting in reduction of the quantum simulation cost from 16 to 12 (for that part). Further, the second class templates can be applied to the pairs of gates 4 and 6, and 7 and 12. However, in order to apply them, the moving rule has to be used to bring the gates together. These two applications lead to the circuit of the form illustrated in Figure 5B. In that circuit, the first and the seventh gates satisfy conditions for the first template and they can be moved together using the moving rule. Application of the first class template to the first and the seventh gates of the circuit in Figure 5B leads to the circuit depicted in Figure 5C. Finally, once no other circuit simplification can be found, the gates are expanded (similarly to the middle circuit in Figure 2) and the templates from the classes 1 and 2 are used to simplify the expanded circuit. This leads to the circuit with quantum simulation cost 96, illustrated in Figure 5D. We have some evidence of further simplification of this circuit once some new templates are introduced and properly applied.

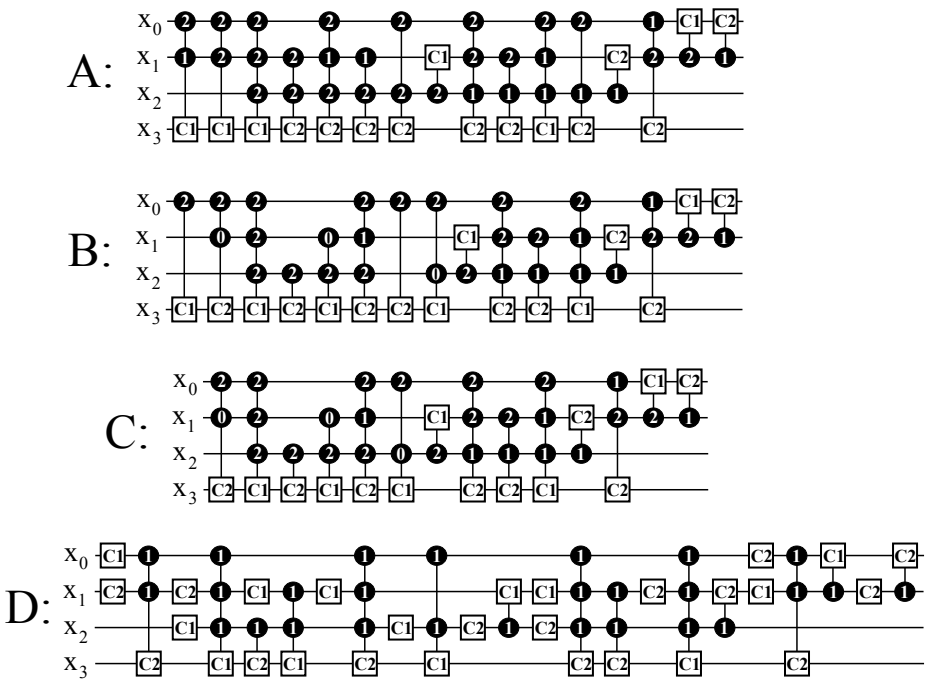

Figure 5: The simplification steps.

9 Conclusion

This paper introduces a simple heuristic algorithm for the synthesis of quantum MVL reversible circuits composed of MVL reversible gates based on the ideas of De Vos *et al.* [18] that are one possible generalization of binary Toffoli gates [17]. While the initial results are quite promising there is considerable need and scope for further research.

In earlier binary work [12], we presented a similar heuristic algorithm and then gave a template-based reduction procedure that can significantly reduce the size of the circuit. We are currently working on templates for the MVL case. Also, while we were able to find quite a reasonable circuit for a ternary full adder as a reversible circuit, work is needed to identify a general procedure for embedding an irreversible specification within a larger reversible specification so that the resulting reversible circuit is minimal, or at least near-minimal. This is an open question for both binary and MVL problems.

The basic reversible gates considered here are only one possible generalization of Toffoli gates, and there are many other MVL reversible gates that could be considered. In particular, we next plan to embed the MVL Fredkin gate [6] into the synthesis algorithm, as its quantum simulation cost does not increase as we go from the binary to MVL case. Mod-sum based reversible gates may also be considered.

The usefulness of this work depends on the efficient realization of the basic reversible gates. Our preliminary investigation shows that the $C1$, $C2$ and N gates and their controlled counterparts will have quite reasonable quantum logic realizations. The D and E type gates would seem to be somewhat more expensive.

Finally, the synthesis algorithm presented here is greedy and heuristic. We are extending the approach to use back-tracking to further improve the quality of the solution. We are also considering an approach which does not necessarily process the input/output patterns in order.

References

[1] IBM's test-tube quantum computer makes history. Technical report, IBM T.J. Watson Research Center, http://researchweb.watson.ibm.com/resources/news/20011219_quantum.shtml, Dec. 2001.

[2] Anas Al-Rabadi. New multiple-valued galois field sum-of-product cascades and lattices for multiple-valued quantum logic synthesis. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 171–182, March 2003.

[3] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[4] C. H. Bennett. Logical reversibility of computation. *IBM J. Research and Development*, 17:525–532, November 1973.

[5] R. Feynman. Quantum mechanical computers. *Optic News*, 11:11–20, 1985.

[6] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.

[7] Richard Hughes et al. Quantum computing roadmap. Technical report, University of California for the National Nuclear Security Administration, of the US Department of Energy, http://qist.lanl.gov/, Dec. 2002.

[8] Mozammel H. A. Khan, Marek A. Perkowski, and Pawel Kerntopf. Multi-output galois field sum of products (gfsop) synthesis with new quantum cascades. In *International Symposium on Multi-Valued Logic*, pages 146–153, May 2003.

[9] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Research and Development*, 5:183–191, 1961.

[10] D. Maslov. *Reversible Logic Synthesis*. PhD thesis, University of New Brunswick, Fredericton, Canada, October 2003.

[11] D. Maslov, G. Dueck, and M.Miller. Simplification of Toffoli networks via templates. In *Symposium on Integrated Circuits and System Design*, pages 53–58, September 2003.

[12] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the Design Automation Conference*, pages 318–323, June 2003.

[13] M. Miller, G. Dueck, and D. Maslov. A synthesis method for MVL reversible logic. In *International Symposium on Multiple-Valued Logic*, Toronto, Canada, May 2004. Accepted.

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

[15] P. Picton. Modified Fredkin gates in logic design. *Microelectronics Journal*, 25:437–441, 1994.

[16] P. Picton. A universal architecture for multiple-valued reversible logic. *MVL Journal*, 5:27–37, 2000.

[17] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.

[18] A. De Vos, B. Raa, and L. Storme. Generating the group of reversible logic gates. *J. of Physics A: Mathematical and General*, 35:7063–7078, 2002.

[19] Victor V. Zhirnov, Ralph K. Kavin, James A. Hutchby, and George I. Bourianoff. Limits to binary logic switch scaling - a gedanken model. *Proceedings of the IEEE*, 91(11):1934–1939, November 2003.

gates	C1, CN	C1, CC1, CN	C1, N, CC1, CN	C1, C2, N, CC1, CC2, CN
0	1	1	1	1
1	4	6	8	12
2	13	31	52	93
3	39	130	280	597
4	115	498	1,342	3,224
5	326	1,777	5,692	15,042
6	897	5,924	20,992	57,951
7	2,395	18,089	63,292	144,039
8	6,107	47,849	128,159	127,056
9	14,660	99,576	118,635	14,750
10	32,268	126,981	23,516	115
11	62,145	58,192	906	
12	96,237	3,795	5	
13	97,705	31		
14	43,902			
15	5,816			
16	243			
17	7			
Avg.	11.97	9.39	8.11	7.16

Table 3: Synthesis of the optimal circuits.

	C2
x	x^+
0	2
1	0
2	1

Table 4: MVL gate $C2$.

		CC2	
x	y	x^+	y^+
0	0	0	0
0	1	0	1
0	2	0	2
1	0	1	2
1	1	1	0
1	2	1	1
2	0	2	0
2	1	2	1
2	2	2	2

Table 5: MVL gate $CC2$.

	D	**E**
x	x^+	x^+
0	0	1
1	2	0
2	1	2

Table 6: MVL gates D and E.

transform	**choice**
$0 \to 1$	$C1, E$
$0 \to 2$	$C2, N$
$1 \to 0$	$C2, E$
$1 \to 2$	D
$2 \to 0$	$C1, N$
$2 \to 1$	D

Table 7: Gate application choices.

(a)		**(b)**		**(c)**	
xy	x^+y^+	xy	x^+y^+	xy	x^+y^+
00	21	**22**	21	22	2**2**
01	20	**20**	20	20	20
02	22	**21**	22	21	2**1**
10	12	1**2**	12	12	12
11	10	1**0**	10	10	10
12	11	1**1**	11	11	11
02	02	**02**	02	02	02
21	00	**00**	00	00	00
22	01	**01**	01	01	01

Table 8: Synthesis example.

Gates	**Algorithm** F **only**	**Algorithm** F **and** F^{-1}	**Optimal**
0	1	1	1
1	24	24	24
2	301	315	335
3	2,395	2,593	3,407
4	11,743	12,954	25,255
5	34,755	39,061	114,095
6	72,217	80,699	187,569
7	97,192	103,663	32,173
8	81,978	79,099	21
9	43,886	34,338	
10	14,849	8,612	
11	3,246	1,437	
12	293	84	
Avg.	7.11	6.92	5.60

Table 9: Synthesis of all 2×2 3-valued reversible functions.

x_2	x_1	x_0	*carry*	*sum*
0	0	0	0	0
0	0	1	0	1
0	0	2	0	2
0	1	0	0	1
0	1	1	0	2
0	1	2	1	0
0	2	0	0	2
0	2	1	1	0
0	2	2	1	1
1	0	0	0	1
1	0	1	0	2
1	0	2	1	0
1	1	0	0	2
1	1	1	1	0
1	1	2	1	1
1	2	0	1	0
1	2	1	1	1
1	2	2	1	2
2	0	0	0	2
2	0	1	1	0
2	0	2	1	1
2	1	0	1	0
2	1	1	1	1
2	1	2	1	2
2	2	0	1	1
2	2	1	1	2
2	2	2	2	0

Table 10: 3-valued 3-bit full adder.