Spectral Techniques for Reversible Logic Synthesis

D. Michael Miller

Department of Computer Science
University of Victoria
Victoria, BC, Canada V8w 3P6
mmiller@csr.uvic.ca

Gerhard W. Dueck

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada E3B 5A3
gdueck@unb.ca

ABSTRACT

Reversible circuits can lead to low-power CMOS implementations and are also of interest in optical and quantum computing. In this paper, we consider the synthesis of reversible logic assuming a generalized Toffoli gate. We make use of Rademacher-Walsh spectral techniques and in particular a spectral measure of function complexity used as a metric in guiding the search for a solution. The synthesis procedure introduced develops the circuit from inputs to outputs and from outputs to inputs simultaneously taking advantage of the best translation available at each step. No backtracking or look-ahead is used. Preliminary results are given for reversible and nonreversible functions together with several ideas for future work.

1. INTRODUCTION

A digital circuit is reversible if it maps each input pattern to a unique output pattern. Landauer [9] proved that traditional irreversible gates lead to power dissipation in a circuit regardless of its implementation. Bennett [1] showed that for power not to be dissipated it is necessary that the circuit be build from reversible gates. Reversible circuits are of interest because of their potential application in low-power CMOS design, quantum computation [12] and optical computing.

Here we consider the synthesis of a reversible circuit as a composition of reversible logic gates. We do not elaborate on technology issues but do make the following assumptions:

(i) fan-out is not permitted;

(ii) loops are not permitted; and

(ii) permutation of connections between gates is permitted.

We assume a generalized Toffoli gate which encompasses the NOT, Feynman [4], and 3*3 and 4*4 Toffoli [19] gates. Fredkin [5] gates are not handled explicitly but can be identified as a pair of identical Feynman gates with an appropriate intervening Toffoli gate. The work presented here is an extension to the work presented by the first author in [13].

Synthesis of reversible logic is significantly different from conventional logic 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. 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, a significant improvement to the method in [13] which worked only from the outputs toward the inputs. In addition, as will be shown by example, it is advantageous to synthesize a circuit for a given reversible specification and also for the inverse specification taking the solution to be the simpler of the two results.

The paper is organized as follows. Section 2 provides the necessary background on reversible logic, the Researcher-Walsh spectral domain, and the spectral measure of function complexity used in this work. Section 3 describes output and input translation and presents the synthesis method. Reversible and irreversible examples are presented in Sections 4 and 5, respectively. The paper concludes with observations and suggestions for further work in Section 6.

2. PRELIMINARIES

2.1 Reversible Logic Gates

A reversible logic gate is a *k*-input, *k*-output (denoted *k***k*) device that maps each possible input pattern to a unique output pattern. For the gates considered, not only is the circuit reversible, the forward and reverse mappings are identical. Many reversible logic gates have been studied in the literature [2][4][5] [8][10][15][19]. Table 1 defines the reversible gates commonly considered in the literature.

The bi-directional nature of these gates is emphasized by using conventional quantum logic notation [14] where the same labels are used on both sides of the gate. We use + to denote the transformed side rather than a prime as often used in the literature in order to avoid any confusion with complementation.

One can readily verify each of these gates is reversible. Feynman and 3*3 Toffoli gates transform a single variable while the Fredkin gate transform a pair of outputs.

Gate Type	Functionality	Gate Notation
1*1 Not	$x^+ = \overline{x}$	NOT(*x*)
2*2 Feynman [4]	$x^+ = x$ $y^+ = x \oplus y$	FEY(*x*,*y*)
3*3 Toffoli [19]	$x^+ = x$ $y^+ = y$ $z^+ = xy \oplus z$	TOF3(*x*,*y*,*z*)
3*3 Fredkin [5]	$x^+ = x$ $y^+ = \overline{x}y \oplus xz$ $z^+ = \overline{x}z \oplus xy$	FRE(*x*,*y*,*z*)

Table 1: Reversible Logic Gates.

The obvious transformations apply regarding NOT and FEY. Further:

$$\mathrm{FRE}(x,y,z) = \mathrm{FEY}(y,z)\,\mathrm{TOF3}(x,z,y)\,\mathrm{FEY}(y,z) \quad (1)$$

from which it follows that

$$\mathrm{TOF3}(x,y,z) = \mathrm{FEY}(z,y)\,\mathrm{FRE}(x,z,y)\,\mathrm{FEY}(z,y) \quad (2)$$

The Toffoli gate can be generalized to the $n * n$ case in the obvious way giving

$$\begin{aligned} &TOFn(x_1,x_2,...,x_{n-1},y) \\ &x_i^+ = x_i, y^+ = x_1x_2...x_{n-1} \oplus y \end{aligned} \quad (3)$$

From Table 1, we can see a Feynman gate is a TOF2 and a NOT gate can be interpreted as a TOF1 by treating the AND of an empty set of variables as 1 and recalling $1 \oplus x = \overline{x}$. Equation (1) shows a Fredkin gate can be realized as a sequence of three generalized Toffoli gates. Hence we shall concentrate on the use of generalized TOFn gates in our synthesis method.

Note that higher order Toffoli gates can be expressed as combinations of Toffoli gates with fewer inputs by introducing constant inputs. For example,

$$\mathrm{TOF4}(w,x,y,z) = \mathrm{TOF3}(w,x,e)\ \mathrm{TOF3}(y,e,z) \quad (4)$$

where e is constant 0 as input to the left TOF3.

We assume complementation is internal to the generalized Toffoli gate and will for example write TOF3(*a′,b′,c*) to mean $c^+ = a'b' \oplus c$ with *a* and *b* unaffected as they pass through the gate. Our synthesis procedure does identify some complementations which can not be absorbed into other gates. These we shall write as TOF1 gates.

2.2 Rademacher-Walsh Spectral Domain

A completely-specified Boolean function $f(x_n,...,x_2,x_1)$ is defined by a column vector of 2^n 0's and 1's denoted $\mathbf{F}$. The Rademacher-Walsh spectrum [6] of the function is given by

$$\mathbf{R} = \mathbf{T}^n\mathbf{F} \quad (5)$$

where the transform matrix $\mathbf{T}^n$ is a Hadamard matrix defined as

$$\begin{aligned} \mathbf{T}^0 &= [1] \\ \mathbf{T}^p &= \begin{bmatrix} \mathbf{T}^{p-1} & \mathbf{T}^{p-1} \\ \mathbf{T}^{p-1} & -\mathbf{T}^{p-1} \end{bmatrix} \end{aligned} \quad (6)$$

Taking +1 as logic 0 and –1 as logic 1, each row of the transform matrix can be seen to correspond to the truth vector of the exclusive-OR (EXOR) of a subset of $x_1,x_2...x_n$. Each spectral coefficient thus measures the correlation of $\mathbf{F}$ to a particular EXOR function and the spectral coefficients are identified by subscripts indicating the variables involved, *e.g.* for $n = 3$,

$$\mathbf{R} = \left[r_0\ r_1\ r_2\ r_{1,2}\ r_3\ r_{1,3}\ r_{2,3}\ r_{1,2,3} \right] \quad (7)$$

r_0 denotes the EXOR of no variables, *i.e.* the constant 0 function and can be seen to simply count the number of 1's in $\mathbf{F}$. All the other coefficients take values in the range -2^{n-1} denoting perfect agreement with the corresponding EXOR function and $+2^{n-1}$ denoting perfect agreement with the complement of that EXOR function.

The coefficients $r_1,r_2,...,r_n$ are termed the first-order coefficients. Each measures correlation to a single variable. These are of particular importance to our method.

Consider two functions, f and g with spectra $\mathbf{R}^{\mathrm{f}}$ and $\mathbf{R}^{\mathrm{g}}$, respectively. Boolean operations can be performed directly in the spectral domain [6] as shown in Table 1.

NOT	$r_0^{\overline{\mathrm{f}}} = 2^n - r_0^{\mathrm{f}};\ r_v^{\overline{\mathrm{f}}} = -r_v^{\mathrm{f}}\ \forall v \neq 0$
AND	$r_v^{\mathrm{fg}} = \sum_{u=0}^{2^n-1} r_v^{\mathrm{f}} \times r_{v\oplus u}^{\mathrm{g}}\ \forall v$ ($v \oplus u$ is bit-wise $\oplus$)
OR	$\mathbf{R}^{\mathrm{f+g}} = \mathbf{R}^{\mathrm{f}} + \mathbf{R}^{\mathrm{g}} - \mathbf{R}^{\mathrm{fg}}$
EXOR	$\mathbf{R}^{\mathrm{f}\oplus\mathrm{g}} = \mathbf{R}^{\mathrm{f}} + \mathbf{R}^{\mathrm{g}} - 2\mathbf{R}^{\mathrm{fg}}$

Table 2: Logic Computations in the Spectral Domain.

2.3 Function Complexity

One simple measure of function complexity is a count of the number of adjacent 0's and adjacent 1's on its Karnaugh map [7]. It has been shown [6], that this count is given by

$$C(f) = \frac{1}{2}\left(n2^n - \frac{1}{2^{n-2}} \sum_{v=0}^{2^n-1} \|v\| \mathbf{r}_v^2 \right) \quad (8)$$

where $\|v\|$ is the number of 1's in the binary expansion of v.

$Z(\mathbf{R})$, the number of zero coefficients in $\mathbf{R}$, is a simple measure of the complexity of the spectrum of a function but is not a direct measure of function complexity since all EXOR functions, including single variables and their complements, have $2^n - 2$ zero-valued coefficients. However, all such functions are equivalent under linear translation [6][7], and hence amenable to implementation using Feynman gates, so $Z(\mathbf{R})$ is of interest.

In the synthesis procedure described below, we use the complexity metric

$$D(f) = n2^n Z(\mathbf{R}) + C(f) \quad (9)$$

$D(f)$ gives a higher value the greater the number of zero-valued spectral coefficients, and when that measure is equal, to functions with higher adjacency count. Single variables and their complements yield the maximum value of $D(f)$ for a given n although they are not unique in that regard.

3. SYNTHESIS

The synthesis approach presented here is an enhancement to the method presented by the first author in [13]. In particular, the earlier method developed the circuit from the outputs towards the inputs. Here we develop the circuit in both directions.

3.1 Output Translation

An *output translation* is the application of a TOFn gate across a set of functions (outputs). In general, this can be expressed as

$$\mathrm{TOF}\, n(f_\alpha, f_{\alpha_2}, ..., f_{\alpha_{n-1}}, f_\beta) \quad (10)$$

The single output f_β is affected. For example, Table 3 shows the application of the output translation $\mathrm{TOF}\, 3(f_0, f_1, f_2)$ which results in $f_2^+ = f_0 f_1 \oplus f_2$.

c b a	$f_0 f_1 f_2$
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 1 0
1 1 0	1 0 1
1 1 1	1 1 1

(a)

c b a	$f_0 f_1 f_2^+$
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	***1 1 1***
1 1 0	1 0 1
1 1 1	***1 1 0***

(b)

Table 3: (a) before translation (b) after output translation.

This simple translation interchanges two output patterns (emphasized in bold italics). In general, an output translation will interchange blocks of rows. The reversibility of the specification is clearly preserved. The synthesis method in [13] employed only this form of translation. In our method the translation is performed directly on the function spectra.

3.2 Input Translation

An *input translation* is similar but applies to the input side of the specification. The general form is

$$\mathrm{TOF}\, n(x_{\alpha_1}, x_{\alpha_2}, ..., x_{\alpha_{n-1}}, x_\beta) \quad (11)$$

Only input x_β is affected. Table 4 illustrates application of the input translation $\mathrm{TOF}\, 2(b, c)$ resulting in $c^+ = c \oplus b$.

c b a	$f_0 f_1 f_2$
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	***1 1 0***
0 1 1	***1 1 1***
1 0 0	1 0 0
1 0 1	1 1 0
1 1 0	***0 1 0***
1 1 1	***0 1 1***

(a)

c^+ b a	$f_0 f_1 f_2$
0 0 0	0 0 0
0 0 1	0 0 1
1 1 0	0 1 0
1 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 1 0
0 1 0	1 0 1
0 1 1	1 1 1

(b)

c^+ b a	$f_0 f_1 f_2$
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	***1 0 1***
0 1 1	***1 1 1***
1 0 0	1 0 0
1 0 1	1 1 0
1 1 0	***0 1 0***
1 1 1	***0 1 1***

(c)

Table 4: (a) before translation (b) after input translation (c) after rearrangement.

Table 4 (c) shows the result after rearranging the input patterns into standard order. From this we see that an input translation in fact interchanges blocks of output patterns. In general, as shown, the movement of output patterns resulting from an input translation cannot be accomplished by an output translation. Also note that an input translation will generally affect more than one output, and often all outputs, whereas an output translation always affects a single output.

Input translations can be carried out directly in the spectral domain but except for TOF1 and TOF2, the operations are rather complex. Our current implementation performs the translation in the function domain and then transforms the results to spectra.

3.3 Synthesis Method

Given a reversible logic specification expressed as a system of Boolean functions $f_i(x_n, ..., x_2, x_1), 1 \le i \le n$, our method first transforms each function to the spectral domain giving the spectra $\mathbf{R}_i, 1 \le i \le n.$.

The first phase of our method is the iterative application of the following procedure which for each iteration identifies a single generalized Toffoli gate and input complementation pattern. The identified gate corresponds to either an input or an output translation, whichever is found to be best in terms of improving the complexity measure *D*. Computations for output translation are carried out in the spectral domain using the rules in Table 2. Computations for input translations require conversion between the spectral and functional domains.

Procedure

Input: Spectra $\mathbf{R}_i, 1 \le i \le n$.

Output: One generalized Toffoli gate and input permutation pattern, and transformed spectra $\mathbf{R}_i^+, 1 \le i \le n$.

Process:

(a) Examine the effect of each input translation and select the translation (generalized Toffoli gate and input complementation pattern) that results in the maximum positive change in $D(f_i)$ summed across all the output functions with no negative change to any $D(f_i)$. In the case of a tie, choose the one with fewest gate inputs and if there is more than one with maximum positive change and the same number of inputs, choose the one with the minimum number of input complementations. If there is still a tie, the first encountered is arbitrarily chosen. Note that complementation of the variable affected by a Toffoli gate is never considered.

(b) Examine the effect of each available output translation and select the translation (generalized Toffoli gate and input complementation pattern) that results in the maximum positive change in $D(f_i)$ for the single output function affected by the translation. Ties are resolved in the same manner as for input translations. Note that output translations are not considered that would affect an output that has already been translated to a single variable or its complement. Once again, complementation of the variable affected by the Toffoli gate is not considered.

(c) If neither (a) or (b) identifies a translation, the procedure terminates in error. Otherwise choose the translation from (a) or (b) that results in the greatest improvement. In the case of a tie, the input translation is chosen.

(d) If an output translation is chosen in (c), $\mathbf{R}_j^+ = \mathbf{R}_j$, $j \ne i$, where f_i is the output affected by the selected gate. $\mathbf{R}_i^+$ is computed based on the chosen translation directly in the spectral domain.

(e) If an input translation is chosen in (c), each of the $\mathbf{R}_i$ is transformed to the functional domain, the functional specifications are rearranged according to the input translation, and the $\mathbf{R}_i^+$ are then found by transforming the permuted functional specifications.

The synthesis process is complete when the spectra $\mathbf{R}_i$ each represent a unique variable or its complement. At this point we have an ordered list of possible interleaved input and output translations, each with an associated generalized Toffoli gate and input complementation pattern.

In the second phase of our procedure, the circuit is produced as an ordered sequence of generalized Toffoli gates as follows:

(a) Each output function that has been translated to the complement of an input variable requires a TOF1 (NOT) gate as a final output translation.

(b) The output permutation $p_i, 1 \le i \le n$, is such that x_{p_i} is the variable, or complemented variable, identified by $\mathbf{R}_i$.

(c) The circuit begins with the Toffoli gates from input translations listed in the order they were identified.

(d) The circuits concludes with the Toffoli gates from output translations listed in ***reverse*** order to the order in which they were identified. This set of Toffoli gates begins with any TOF1 gates identified in (a). The variable labels for the Toffoli gates associated with output translations including the TOF1 gates are relabeled according to the permutation identified in (b).

(e) The identified circuit produces the originally specified outputs in the permutation order identified in (b).

(f) As a final step, Fredkin gates can be inserted by applying (1). Other optimizations can be applied such as inverter reduction.

The circuit is produced as a sequence of Toffoli gates from the input side to the output side. Of course, applying the sequence in reverse transforms the output side to the input side.

The synthesis procedure described here is a significant extension to the one described in [13] as it uses input translation as well as output translation. In addition, the procedure has been extended to generalized Toffoli gates. The work in [13] considered only up to TOF4 gates. Finally, the procedure described here uses a different rule for breaking ties. Preference is here given to maximizing the improvement in the function complexity metric whereas in [13] preference was given to minimizing the gate width. Experimentation has shown the scheme used here in general yields better results.

4. REVERSIBLE EXAMPLES

For each example, the specification is given as an ordered set of decimal numbers which define the truth table specification of the reversible logic problem to be realized. To illustrate, the specification for Example 1 defines a Fredkin gate as specified in Table 5. The circuit is given as an ordered sequence of reversible gates. Read from left to right they transform the left side of the specification to the right side. The output permutation is not noted if it is the identity.

Example 1: Verification of realizing a Fredkin gate using two Feynman gates and a Toffoli gate.

Specification: [0,1,2,3,4,6,5,7] (corresponds to Table 5)

Circuit: TOF2(b,a) TOF3(c,a,b) TOF2(b,a)

c b a	c^+ b^+ a^+
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 1 0
1 1 0	1 0 1
1 1 1	1 1 1

Table 5 Fredkin Gate Specification.

Example 2: This is a second example of the interchange of two positions in the specification. Note the analogous structure to that of the Fredkin gate realization – the right pair of TOF2 gates can be reversed to better show the expected symmetry. The circuit given by our method is identical to a solution provided by Perkowski [16].

Specification: [0,1,2,4,3,5,6,7]
Circuit: TOF2(*c,b*) TOF2(*c,a*) TOF3(*b,a,c*) TOF2(*c,b*) TOF2(*c,a*)

Example 3: The four input extension of Example 2. The right three TOF2 gates can be permuted to better show the symmetry.

Specification: [0,1,2,3,4,5,6,8,7,9,10,11,12,13,14,15]
Circuit: TOF2(*d,c*) TOF2(*d,b*) TOF2(*d,a*) TOF4(*c,b,a,d*) TOF2(*d,c*) TOF2(*d,b*) TOF2(*d,a*)

Example 4: This is increment $\bmod 2^n$ for $n = 3$.

Specification: [1,2,3,4,5,6,7,0]
Circuit: TOF3(*b,a,c*) TOF2(*a,b*) TOF1(*a*)

Example 5: This is the 4 input extension of Example 4. Note the generalization to the structure of the solution for the $n = 3$ case. The structure generalizes in the same manner for higher values of *n*.

Specification: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0]
Circuit: TOF4(*c,b,a,d*) TOF3(*b,a,c*) TOF2(*a,b*) TOF1(*a*)

Example 6: This example is taken from [17].

Specification: [3,11,2,10,0,7,1,6,15,8,14,9,13,5,12,4]

Circuit: TOF1(*c*) TOF1(*b*) TOF2(*a,d*) TOF2(*c,d*) TOF3(*a',d,c*) TOF2(*c,d*) TOF2(*c,a*) TOF3(*a',d',b*)

The three underlined gates in the above solution can be replaced with the single Fredkin gate FRE(*a',c,d*). The outputs of the circuit are the permuted order (*c,a,d,b*). Note that (*d,c,b,a*) is the standard unpermuted order. The solution is comparable to the solution given in [17].

Example 7: This is the inverse to the specification in Example 6. This shows the importance of applying our synthesis method to both a specification and the corresponding inverse (except when the specification is self-inverse) and choosing the better of the results.

Specification: [4,6,2,0,15,13,7,5,9,11,3,1,14,12,10,8]

Circuit: TOF3(*c',b',a*) TOF2(*c,b*) TOF2(*c,d*) TOF1(*b*) TOF1(*a*) TOF3(*b,d,c*) TOF2(*c,b*)

In this case, the output permutation is (*c,b,a,d*).

Comparing the solution for Examples 6 and 7 shows the importance of applying the synthesis method to a problem and its inverse. The resulting circuit can be quite different. One can of course choose between the solution to a problem or its inverse, since reading a solution in reverse gives a circuit for the inverse problem due to the reversibility of the gates. For example, reading the solution for example 6 from right to left is a solution for Example 7, and reading the solution for Example 7 from right to left is a solution for Example 6.

5. IRREVERSIBLE EXAMPLES

An arbitrary combinational circuit composed of irreversible gates can be mapped to a reversible circuit typically with the addition of some number of constant inputs and 'garbage' outputs [14]. Here our interest is to synthesize a reversible circuit from an irreversible specification and not the transformation of a irreversible circuit to a reversible one. To apply our synthesis method, an irreversible specification must first be transformed to a reversible one. We assume the given specification is totally-specified.

5.1 Single-Output Functions

A single-output function f involving input variables $x_1, x_2, ..., x_n$ is transformed to a specification with $n + 1$ inputs and $n + 1$ outputs by:

i) adding a new input variable x_{n+1},

ii) replacing the output *f* by $f \oplus x_{n+1}$,

iii) adding *n* outputs each equal to one of the original inputs $x_1, x_2, ..., x_n$.

It is easily verified that the specification constructed in this way assigns a unique output pattern to each input pattern and is therefore reversible. By setting $x_{n+1} = 0$ on input, the original output *f* is realized.

Example 8: This procedure for transforming a single-output function is illustrated for the simple example of the 2-input AND function in Table 6. The resulting circuit is the single gate TOF3(*b,a,c*) which realizes the AND of *a* and *b* when *c* is 0 on input.

b a	f
0 0	0
0 1	0
1 0	0
1 1	1

(a)

c b a	c^+ b^+ a^+
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	1 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	0 1 1

(b)

Table 6: (a) 2-input AND (b) reversible specification derived from 2-input AND.

Example 9: The *two of five checker* function has five inputs and a single output which is 1 if, and only if, two of the inputs are 1. Transforming this function to a reversible specification as described above and then applying our synthesis procedure yields the circuit:

TOF3(*b',a',f*) TOF4(*d',c',a,f*) TOF4(*e',c',b',f*)
TOF6(*e,d',c',b,a,f*) TOF6(*e',d',c,b',a,f*) TOF6(*e',d,c',b,a',f*)
TOF6(*e,d,c,b',a',f*) TOF5(*e',d',c,a',f*) TOF5(*e,d',c',a',f*)

with output permutation (*f,e,d,c,b,a*). Variable *f* is the desired output and should be set to 0 on input. This solution uses two more Toffoli gates than the solution found by Dueck and Maslov [3]. Their algorithm uses exhaustive evaluation at each stage including two-step look-ahead to choose the best generalized Toffoli gate.

Example 10: To illustrate this process is equally effective for a non-symmetric function, consider the 5-input, single output function which is 1 for the five input patterns *e'd'c'b'a*, *e'd'c'ba*, *e'd'cba*, *e'dcba* and *edcba*. Our approach identifies the simple circuit

TOF6(*e,d,c,b,a,f*) TOF5(*e',c,b,a,f*) TOF5(*e',d',c',a,f*)

where again variable *f* is the desired output and should be set to 0 on input.

5.2 Multiple-Output Functions

The approach described above or an extension to it is applied for multiple-output functions. In this case, the number of outputs to be added is at least $\lceil \log_2 m \rceil$ where *m* is the maximum number of times a single output pattern appears in the given specification [11]. Additional outputs and also additional inputs may be required to ensure the derived specification has the same number of inputs and outputs, a requirement for it to be reversible. The transformation of the given specification must be done in such a way as to yield a reversible specification without introducing undue complexity.

Example 11: Table 7 is a reversible specification derived from a full adder specification shown in bold italics. Two outputs must be added, as the output patterns 01 and 10 each appear 3 times in the specification of the full adder.

The resulting circuit is

TOF2(*b,c*) TOF2(*a,c*) TOF1(*d*) TOF2(*c,d*) TOF4(*c',b',a',d*)
TOF4(*c,b,a,d*)

which can be simplified by replacing TOF1(*d*) TOF2(*c,d*) by TOF2(*c',d*). Applying our synthesis method to the inverse of the specification in Table 7, yields the similar complexity circuit:

TOF2(*b,c*) TOF2(*a,c*) TOF2(*c,d*) TOF4(*c,b',a',d*)
TOF4(*c',b,a,d*)

d	c	b	a	d^+	c^+	b^+	a^+
0	***0***	***0***	***0***	***0***	***0***	0	0
0	***0***	***0***	***1***	***0***	***1***	0	1
0	***0***	***1***	***0***	***0***	***1***	1	0
0	***0***	***1***	***1***	***1***	***0***	1	1
0	***1***	***0***	***0***	***0***	***1***	0	0
0	***1***	***0***	***1***	***1***	***0***	0	1
0	***1***	***1***	***0***	***1***	***0***	1	0
0	***1***	***1***	***1***	***1***	***1***	1	1
1	0	0	0	1	0	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	0
1	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
1	1	0	1	0	0	0	1
1	1	1	0	0	0	1	0
1	1	1	1	0	1	1	1

Table 7: Reversible specification derived from the full adder (bold italics).

The simpler circuit

TOF2(*b,a*) TOF2(*a,c*) TOF3(*c,a,d*) TOF3(*b',a',d*) TOF1(*d*)

is found by changing the specification in Table 7 so that $a^+ = a \oplus b$. This definition for a^+ is taking into account that $a \oplus b$ is a useful subfunction in the realization of the sum and carry functions of the full adder. Knowledge of the decomposition structure of the target functions is of considerable benefit in choosing a reversible specification.

Applying our synthesis approach to the inverse of the latter specification yields:

TOF2(*a,c*) TOF2(*b,a*) TOF2(*c,d*) TOF4(*c,b',a',d*)
TOF4(*c',b,a,d)*

which is more complex. This latter circuit is interesting to compare to the circuit found from considering the inverse of the original specification in Table 7.

Example 11: As a final example we consider the benchmark function RD53. This function has 5 inputs and 3 outputs. The outputs are the binary encoding of the weight of the input pattern *i.e.* the number of 1's in the input pattern. For example, input 00000 yields output 000, input 00100 yields output 001 and input 11111 yields output 101.

The maximum output pattern multiplicity is 10 so at least 4 garbage outputs must be added giving a total of at least 7 outputs. That in turn requires two inputs be added. We thus have a specification with inputs g, f, e, d, c, b, a and outputs $g^+, f^+, e^+, d^+, c^+, b^+, a^+$ where e, d, c, b, a are the original inputs. Let h_2, h_1, h_0 be the original outputs with h_2 the most significant bit of the binary representation of the weight.

A reversible specification can be derived using the principles outlined above by setting

$$g^+ = h_2 \oplus g, f^+ = h_1 \oplus f, e^+ = h_0,$$
$$d^+ = e, c^+ = d, b^+ = c, a^+ = b. \quad (12)$$

Setting $g = f = 0$ yields the desired outputs. Applying our synthesis method to this specification yields the circuit

TOF4(*e,b,a,g*) TOF2(*e,a*) TOF2(*d,a*) TOF2(*c,a*) TOF2(*b,a*)
TOF1(*f*) TOF6(*a,e',d',c',b,f*) TOF6(*a',e,d,c',b',f*
TOF5(*a,e',d',b',f*) TOF4(*a',e,d,f*) TOF2(*a,f*) TOF6(*a',e,d',c,b,f*)
TOF6(*a,e',d,c',b',f*) TOF6(*a',e',d,c,b,f*) TOF6(*a,e,d',c',b',f*)
TOF2(*a,f*) TOF6(*a',e',d',c',b',f*) TOF6(*a,e,d,c,b,*
TOF6(*a',e',d,c,b,g*) TOF6(*a,e,d',c',b,g*) TOF5(*a',e,d,c,g*)

with output permutation (*g,f,a,e,d,c,b*). The underlined TOF1 gate can be removed by replacing the underlined TOF2 gate by TOF2(*a',f*).

The sequence TOF2(*e,a*) TOF2(*d,a*) TOF2(*c,a*) TOF2(*b,a*) in the above circuit is an indication that parity plays an important role in RD53. Changing the specification in (12) so that $d^+ = e \oplus d \oplus c \oplus b$ and applying our synthesis procedure yields a circuit with 15 generalized Toffoli gates. That circuit shows some further parity dependency. This led us to consider the following reversible specification:

$$g^+ = h_2 \oplus g, f^+ = h_1 \oplus f, e^+ = h_0,$$
$$d^+ = e \oplus d \oplus c \oplus b, c^+ = d \oplus c \oplus b, \qquad (13)$$
$$b^+ = c \oplus b, a^+ = b.$$

Applying our synthesis algorithm yields the circuit

TOF5(*e,d,c,a,g*) TOF2(*b,c*) TOF2(*c,d*) TOF2(*d,e*) TOF2(*e,a*)
TOF3(*e,a,f*) TOF3(*c',b,f*) TOF3(*d',c,f*) TOF3(*e',d',f*) TOF1(*f*)
TOF5(*a',e,d',b,g*) TOF5(*a',d,c',b,g*)

with output permutation (*g,f,a,e,d,c,b*). This circuit requires 12 Toffoli gates as opposed to 20 for the initial reversible specification. Far fewer input complementations are needed and the width of the required gates is greatly reduced. Note that the TOF1 gate can be removed by complementing f in (13). This assumes a constant 1 can be used as an input in the final circuit implementation. The circuit found by our method in that case uses 11 Toffoli gates, whereas the method in [3] requires 13 Toffoli gates with higher input counts.

This example clearly shows the benefit of taking properties of the target functions and an initial circuit realization into account when constructing and refining the reversible specification.

6. CONCLUSION

Results to date show the spectral-based synthesis method does indeed have promise. We are currently developing a formal proof that the method will terminate giving a circuit for any completely-specified reversible specification.

The preliminary implementation of our synthesis method is in C and represents functions and their spectra as vectors of length 2^n although fast transform methods [6][18] are used in the implementation rather than the matrix transformation method. Execution time on a 750 MHz PC is reasonable for the examples shown, about 3 minutes for RD53. A decision diagram implementation is under development so the method can be applied to larger problems. Preliminary work has shown that input and output translations can be carried out directly on functional or spectral decision diagrams. In particular, input translations can be performed as local translations on a decision diagram provided the variables involved are adjacent.

While our synthesis method does not require extensive look-ahead or back-tracking techniques, it does require the full consideration of the possible Toffoli gates and input complementation patterns at each stage of the synthesis. We are currently investigating spectral criteria to reduce this searching.

Lastly, we are examining how to formalize the transformation of a nonreversible specification to a reversible one. In particular, we are examining how spectral criteria that may aid this process.

REFERENCES

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

[2] De Vos, A., "Towards Reversible Digital Computers," *Proc. European Conf. Circuit Theory & Design*, 1997, pp. 923-931.

[3] Dueck, G. W., and D. Maslov, "Reversible Function Synthesis with Minimum Garbage Outputs," submitted to RM-2003.

[4] Feynman, R., "Quantum Mechanical Computers," *Optics News*,**11**, 1985, pp. 11-20.

[5] Fredkin, E., and T. Toffoli, "Conservative Logic," *International Jour. Theoretical Physics*, 1982, pp. 219-253.

[6] Hurst, S. L., D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press, 1985.

[7] Karpovsky, M. G., *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley and Sons, 1976.

[8] Kerntopf, P., "On Efficiency of Reversible Logic (3,3) Gates," *Proc. 7th Intl. Conf. MIXDES*, 2000, pp. 185-190.

[9] Landauer, R., "Irreversibility and Heat Generation in the Computational Process," *IBM Journal of Research and Development*, **5**, 1961, pp. 183-191.

[10] Margolus, N., *Physics and Computation*, Ph. D. Thesis, Massachusetts Institute of Technology, 1988.

[11] Maslov, D., and G. W. Dueck, "Garbage in Reversible Designs of Multiple-Output Functions," submitted to RM-2003.

[12] Milburn, Gerard J., *The Feynman Processor*, Perseus Books, 1998.

[13] Miller, D. M., "Spectral and Two-Place Decomposition Techniques in Reversible Logic," *Proc. Midwest Symposium on Circuits and Systems,* on CD-ROM, August 2002.

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

[15] Peres, A., "Reversible Logic and Quantum Computers," *Physical Review A*, **32**, 1985, pp. 3266-3276.

[16] Perkowski, M. Private communication.

[17] Perkowski, M., *et al*., "A General Decomposition for Reversible Logic," *Proc. Fifth Reed-Muller Workshop*, 2001, pp. 119-138.

[18] Thornton, M. A., R. Drechsler and D. M. Miller, *Spectral Techniques in VLSI CAD*, Kluwer, 2002.

[19] Toffoli, T., "Reversible Computing," in *Automata, Languages and Programming*, Springer-Verlag, pp. 632-644, 1980.

Acknowledgements: This work was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada. This work was completed while the first author was on sabbatical at the University of New Brunswick.