Technical Report: Considerations for Determining a Classification Scheme for Reversible Boolean Functions

TR-CSJR2-2007
J. E. Rice
University of Lethbridge
j.rice@uleth.ca

1 Introduction

For many years now industry and researchers alike have relied on Moore's law, which commonly states that the complexity of minimum component costs has, and will continue to, increase at a rate of approximately a factor of two per year. The assumption that goes along with this "law" is that the complexity referred to is poportional to the number of transistors. In other words, electronics components will be able to do twice as much, or have twice as many transistors in the same area every year[1]. So far, Moore's law has held true and we enjoy the use of minuscule devices in toasters, cell phones, laptops, PDAs, *etc.* However groups like the Semiconductor Industries Association (SIA), which produces yearly a study called the International Technology Roadmap for Semiconductors (ITRS) [1], identify some alarming trends: the ITRS report shows many areas where technological breakthroughs will be needed to ensure continued progress. The report even seems to indicate that the experts are unsure how the previous scaling factor described by Moore can continue even a decade into the future.

Reversible logic may be the solution to the brick wall we are facing. Frank [10] states

> ... computer based mainly on reversible logic operations can reuse a fraction of the signal energy that theoretically can approach arbitrarily near to 100% as the quality of the hardware is improved...

This is not a new idea; in 1961 Landuaer published the first paper linking reversible logic to lower power dissipation [21], and in 1973 Bennett showed that for power *not* to be dissipated it is necessary that a binary circuit be constructed from reversible gates [3]. As a practical example, De Benedictis discusses how Landauer's limit impacts supercomputing [5].

Reversible logic also has connections to the new "hot" area of quantum computing. and in fact reversible circuits can be viewed as a special case of quanutm circuits [26]. As Shende *et al.* note,

> 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 [40].

Although the area of reversible logic is not new, it is only now that it seems to be gaining interest, and only recently that techniques for actually synthesizing circuits have been researched. This report does not address technology considerations, although the reader is directed to works such as [46, 9, 20] for discussions in this direction.

[1]Some statements of Moore's law state that the doubling will occur every 18 months, but the general idea is the same.

This report provides the reader with background in the current status of synthesis for reversible logic. We propose that, given the number of approaches being considered, a classification scheme may be useful for a variety of reasons:

- to select a particular synthesis approach,
- to identify some useful properties of a reversible function that could then be leveraged in the synthesis process,
- to identify if a function is L-constructible, where L is some library of (reversible) gates, or
- to leverage similarities between an existing implemented circuit (or subcircuit) and the circuit in question.

2 Background

2.1 Reversible Logic

[40] provides the following definitions:

Definition 2.1 *A gate is reversible if the (Boolean) function it computes is bijective.*

Definition 2.2 *A well-formed reversible logic circuit is an acylic combinational logic circuit in which all gates are reversible, and are interconnected without fanout.*

This definition assumes that the circuit is strictly combinational; considerations for sequential logic are addressed in [32]. In general, a function is reversible if there is a one-to-one and on-to mapping from the inputs to the outputs (and vice versa) of the function. This means it is possible to compute how many reversible functions there are by counting the possible rearrangements of the outputs. Since a reversible function with n inputs will require n outputs as well, there are $2^n!$ possible reversible functions. One might assume that there are fewer reversible functions than irreversible, but in fact this is not the case. Table 1 illustrates how the numbers of Boolean functions and reversible functions increase as n increases. It is of

n	2^{2^n}	$2^n!$
1	4	2
2	16	24
3	64	40,320
4	65,536	$2.09E^{13}$

Table 1: The number of traditional functions as compared to the number of reversible functions.

course assumed that the traditional functions are single-output functions, while reversible functions must (for $n > 1$) have multiple outputs.

Because a reversible gate must be bijective, this means that the traditional inverter is considered reversible, while the traditional AND gate is not. The most commonly used reversible gates include the NOT gate (or the inverter), a SWAP gate, and variations on these. Table 3 lists the behaviour of some reversible gates. The Feynman and Toffoli are in essence extensions of the NOT gate, while the Fredkin extends the behaviour of the SWAP gate. Figure 1 illustrates the symbols used for these gates in this report.

x	$\overline{x}$
0	1
1	0

$x\ y$	$x \cdot y$
0 0	0
0 1	0
1 0	0
1 1	1

Table 2: The traditional inverter gate, which is reversible, and the traditional AND gate, which is irreversible.

gate	behaviour
NOT	$(x) \rightarrow (x \oplus 1)$
Feynman	$(x, y) \rightarrow (x, x \oplus y)$
Toffoli	$(x, y, z) \rightarrow (x, y, xy \oplus z)$
SWAP	$(x, y) \rightarrow (y, x)$
Fredkin	$(x, y, z) \rightarrow (x, z, y)$ *iff* $x = 1$

Table 3: The behaviour of a selection of more commonly used reversible logic gates.

2.2 Terminology

In most cases this report will explain terminology as it is needed; in addition we will try to point out when differing terminology is used for the same or similar concepts by different authors. However a short comment on one point is useful: although we refer to "inputs" and "outputs" of a reversible function, this is a misnomer. This is because a reversible function may be implemented using some quantum technology, where wires and traditional technologies cannot be used. It would be more accurate to refer to a starting state and an ending state, and instead of referring to signals or wires in a circuit, it is more accurate to refer to the qubits involved in the circuit. We will assume in this report that any reference to inputs implies the starting state of the qubits for the circuit, and that any reference to output implies the ending state of the circuit's qubits. Of course, since all circuits are reversible the starting and ending states can be reversed, which the reader should also keep in mind while considering these concepts.

2.3 Classification

As shown in Table 1 there are 2^{2^n} possible single-output Boolean functions of n variables. Classification of Boolean functions is a well-known technique in traditional logic for identifying desirable characteristics in functions, and grouping those functions together [14], and can be used to simplify the logic synthesis process [7, 14, 13].

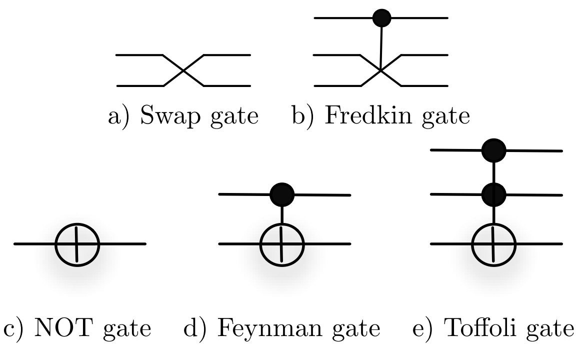


Figure 1: Symbols for some of the more common reversible gates.

There are a number of techniques used in Boolean logic classification. In this work our focus is on on techniques that may be used in digital logic design applications. Clearly it is useful to know whether or not a particular function possesses a certain property, for instance, symmetry. One way to classify functions is according to those properties, and so one might identify symmetrical functions, unate functions, and self-dual functions, to name a few. Of course identification of these properties may not be trivial, and so techniques such as using spectral tests [14] have been suggested.

Another traditional classification technique involves applying so-called “invariant” operations to functions, and assigning all functions that can be generated through the application of these operations to one particular class. The NPN-classes are such an example of these, in which the invariant operations are negation of one or more of the input variables, permutation of two or more of the input variables, and negation of the output of the function.

Transforming a function from the Boolean domain to the spectral domain is another technique used in traditional Boolean logic classification. The application of a transformation matrix such as the Hadamard matrix generates a vector of coefficients as an alternative representation of the function. These coefficients can be grouped, generally according to their magnitude, and any functions with the same magnitude coefficients can be said to reside in the same spectral class. Further details of these techniques can be found in [14].

3 Reversible Synthesis Techniques

As recent researchers have found, synthesizing reversible logic functions is not a trivial process. In fact, nearly all researchers have limited themselves to trials on functions with only 3 inputs. Some of the work in this area includes [22, 40, 25, 30, 2] and [18]. Before identifying properties that are useful, it is necessary to know how the approaches work, and thus a large part of this report is dedicated to an overview of some of these synthesis techniques.

3.1 Maslov *et al.'s* Template Technique

Maslov, Miller and Dueck have published a number of papers leading to what I will refer to here as the template technique. Details for this technique are given in [22]. The technique begins with a truth table describing a reversible function, and produces a cascade of Fredkin and Toffoli gates. The steps are as follows:

1. use NOT gates to transform the first row of outputs in the truth table to $00\cdots0$, and then update all output rows according to the changes.
2. moving down the truth table use the simplest possible (*i.e.* having the fewest control lines) to bring each output pattern to the form of its corresponding input pattern without influencing previous rows in the table.
3. apply the template simplification rule

For example, given the function shown in Table 4 the tranformation process is as follows:

Step 0 No change is required for this step; row 0 is already correct

Step 1 The goal is now to transform the output pattern 011 to match the input pattern 001; one way to do this is to apply the function $y' = y \oplus z$, which matches the Feynman gate, and then apply this function to all of the outputs resulting in the truth table shown in Table 5a).

xyz	$x'y'z'$
000	000
001	011
010	100
011	101
100	110
101	010
110	111
111	001

Table 4: A $n = 3$ reversible function to synthesize.

xyz	$x'y'z'$
000	000
001	001
010	100
011	111
100	110
101	010
110	101
111	011

a)

xyz	$x'y'z'$
000	000
001	001
010	010
011	111
100	110
101	100
110	011
111	101

b)

xyz	$x'y'z'$
000	000
001	001
010	010
011	011
100	110
101	100
110	111
111	101

c)

xyz	$x'y'z'$
000	000
001	001
010	010
011	011
100	100
101	110
110	101
111	111

d)

Table 5: Truth tables corresponding to the steps in the template-based transformation synthesis approach.

Step 2 Row 2 has 100 in the output column and 010 in the input column; swap x' and y' to make them match. The resulting truth table is shown in Table 5b).

Step 3 The next change required is in row 3 wherew there is 111 in the output column and 011 in the input column. The Toffoli gate can be applied to make the required change: $x' = x \oplus yz$. The result is shown in Table 5c).

Step 4 In row 4 the goal is to change 110 to 100; apply $y' = y \oplus x$. The result is shown in Table 5d).

Now all output patterns match the inputs, and the resulting cascade of gates is shown in Figure 2. The final

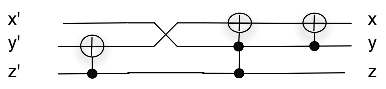


Figure 2: The resulting cascade of gates for the truth table in Table 4.

step is to examine the cascade of gates and replace any series of gates that match a variety of templates determined by the authors (see [22] for details) if such a replacement reduces the number of gates in the series. Templates have been determined by identifying all series of gates which realize the identity function, and by enumerating exhaustively the templates consisting of 1–6 gates. These templates have then been classified to reduce the number of different templates. These classes are based on the functionality of the templates.

3.2 Group Theory and Reversible Logic Synthesis

Although this report does not go into detail, it is worth noting that Van Rentergem *et al.* have performed some interesting analysis of reversible computing using group theory [30]. They also propose some synthesis techniques based on group theory. These techniques are based on decomposing an arbitrary reversible gate (or function) to be implemented into smaller gates with known characteristics. They show that this can be performed with a variety of smaller gate types, and in various ways. This report will not describe all the methods, as they have a similar basis to that published in [31] which is described below.

In [31] these researchers categorized various reversible gates as shown in Table 6. The authors of this

Gate type	p	s
Inverter	2^n	0
Exchanger	$n!$	0
Selective gate	$n!2^n$	0
Simple control gate	$2^{2^{n-1}}$	$(n-1)2^n$
Compact control gate	$2^{n-1}+1$	$4(n-1)$
Arbitrary reversible gate	$2^n!$	$\frac{4}{3}(4^n-3n-1)$

Table 6: Various special types of reversible gates, the number p of different gates of each type and the maximum number s of switches needed in the implementation of each type [31].

work assume that exchanges (swaps) and inverting can be peformed at no cost in their assumed hardware implementation, and so they require 0 switches for each of these gate types. We can see that an arbitrary reversible gate is one that computes any reversible function, and definitions of the other types are given below.

Definition 3.1 *A gate is* selective iff *each of its outputs P_i either equals an input A_j or the inverse $\overline{A_k}$ of an input.*

Definition 3.2 *A gate is a* simple control gate *if*

$$P_i = A_i \text{ for all } i \in \{1,2,\ldots,n-1\}$$
$$P_n = f(A_1,A_2,\ldots,A_{n-1}) \oplus A_n$$

where f denotes an arbitrary Boolean function of $n-1$ Boolean arguments

Definition 3.3 *A gate is a* compact control gate *iff its control function is compact. A compact Boolean function of q Boolean arguments can be written as a combination of OR and AND gates with only q letters (see [31] for futher explanation).*

Definition 3.4 *A SWITCHED gate is defined as a gate for which $P_1 = A_1$ and $P_2,\ldots,P_n$ are the outputs of some reversible gate of width $n-1$ with inputs $A_2,\ldots,A_n$; if $A_1 = 0$ then the $n-1$ reversible gate computes the function g' while if $A_1 = 1$ then the reversible gate computes the function g''.*

The authors point out that two "notorious" gates are SWITCHED gates; the Fredkin gate, where in this case the function g' is a 2-bit follower and the function g'' is a SWAP (or 2-bit exchanger), and the controlled not gate (sometimes called the Feynman gate) where in this case the function g' is a 1-bit follower (unchanged) and g'' is a 1-bit inverter.

This analysis allows the authors to identify that an arbitrary gate g can be built from a SWITCHED gate k_1, an upside-down compact control gate r and a second SWITCHED gate k_2. A picture of this decomposition is shown in Figure 3. The process for identifying which exact gates k_1, g, and k_2 is as follows:

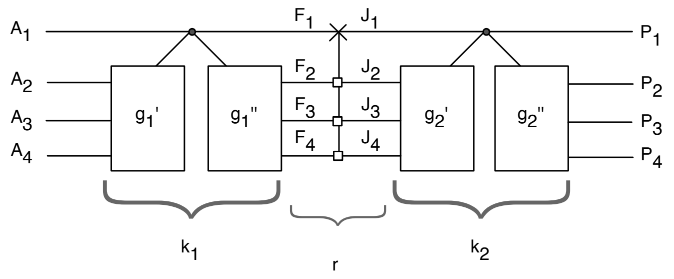


Figure 3: Decomposition of an arbitrary reversible gate (function) [31].

- take the original truth table for the arbitrary gate to be implemented and add columns for F_i and J_i
- set F_1 equal to A_1 and J_1 equal to P_1
- compare column J_1 with column F_1:
  a) for the top half of the truth table (the first 2^{n-1} rows) if $F_1 == J_1$ set the remaining columns $F_2, \ldots, F_n = J_2, \ldots, J_n$ starting with values $0, \ldots, 0$ and incrementing by 1 for each new row
  b) repeat this process for the bottom half of the truth table, again starting with values $0, \ldots, 0$ and incrementing for each row where $F_1 == J_1$
  c) for the first 2^{n-1} rows if $F_1 \neq J_1$ set the remaining columns $F_2, \ldots, F_n = J_2, \ldots, J_n$, continuing where values left off in step a)
  d) repeat this process for the bottom half of the truth table, continuing where values left off in step b).

This results in truth tables for g_1', g_1'', g_2', g_2'' and r. The procedure is recursively applied to the truth tables for g_1', g_1'', g_2', and g_2'' until all of these "subtables" consist of 1-bit followers or 1-bit inverters. Some optimization can be added to this procedure.

3.3 Shende *et al.*

As noted in [40], as early as 1980 Toffoli [42] provided a technique for constructing an arbitrary reversible function in terms of a gate library. The problem with his technique is that the method required a large number of temporary storage channels (sometimes referred to as ancilla bits). The goal of these authors is to avoid the use of temporary storage channels. Figure 4 gives an example of a circuit that requires temporary storage. At this stage it is useful to introduce another notation for specifying a reversible function. Since reversible functions are, by definition, bijective, the outputs are in essence permutations of the inputs. Thus it is possible to represent a reversible function by indicating which rows are swapped to generate the desired permutation. For instance, Table 7 shows a truth table that can be described with the permutations $(2, 3)(6, 7)$. Input rows 2 (010) and 3 (011) are swapped, and input rows 6 (110) and 7 (111) are swapped. Here the inputs are labeled as x, y and z and the outputs as x', y' and z'. Other terminology used also refers to the fact that reversible functions are permutations of the inputs. The set of bijective functions with m indices is denoted S_m and since for n input variables there are 2^n possible combinations (or index values) then the set of reversible functions with n binary inputs is S_{2^n}. Also, since a reversible function must have

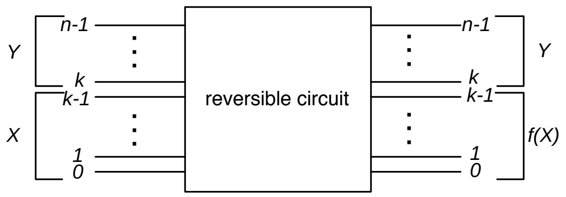


Figure 4: An example of a circuit that requires n inputs and outputs, but for which we are only interested in outputs $0, \ldots, k-1$ [40].

$x\ y\ z$	$x'\ y'\ z'$
000	000
001	001
010	011
011	010
100	100
101	101
110	111
111	110

Table 7: A truth table that can be represented as $(2,3)(6,7)$.

the same number of inputs as outputs, such a function may be referred to as an $n \times n$ circuit, referring to the number of inputs $\times$ the number of outputs.

Definition 3.5 *Let L be a (reversible) gate library. An L-circuit is a circuit composed only of 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.*

Definition 3.6 *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.*

Definition 3.7 *A function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ is* linear *iff $f(x \oplus y) = f(x) \oplus f(y)$ where $\oplus$ denotes bitwise XOR.*

The authors point out that for reversible functions $n = m$ and thus f can be thought of as a square matrix over the two-element field $\mathcal{F}$.

Lemma 3.1 *Every C-constructible permutation computes an invertible linear transformation,* and *every invertible linear transformation is computable by a C-constructible circuit not requiring more than 2^n gates.*

In this case C refers to the CNOT, or controlled not gate:

Definition 3.8 *A k-CNOT gate is a $(k+1) \times (k+1)$ gate which leaves the first k inputs unchanged and inverts the last iff all others are 1; the unchanged lines are referred to as control lines.*

As there are many notations and names used to refer to various reversible gates, it should be noted that the 0-CNOT gate is a NOT gate; the 1-CNOT gate is also referred to as a Feynman gate, and the 2-CNOT gate is a Toffoli gate.

Definition 3.9 *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 referred to as A_n.*

For example, the permutation $(0, 3)(1, 2)$, whose truth table is shown in Table 8a) is even, while the permutation $(1, 2)$, whose truth table is shown in Table 8b) is odd. If a permutation is even then it may not be written as the product of an odd number of transpositions; additionally, half the permutations in S_n are even (for $n > 1$).

a)

xy	$x'y'$
00	11
01	10
10	01
11	00

b)

xy	$x'y'$
00	00
01	10
10	01
11	11

Table 8: a) The truth table for even permutation $(0, 3)(1, 2)$ and b) the truth table for odd permutation $(1, 2)$.

Lemma 3.2 *Any $n \times n$ circuit with no $n \times n$ gates computes an even permutation.*

For instance, in a 2×2 circuit the odd permutation $(1, 2)$ can be computed with a single swap gate which is a 2×2 gate. However in a 3×3 circuit a single 2×2 swap gate computes the even permutation $(1, 2)(5, 6)$. Note that this implies that for $n \geq 4$ every permutation constructed with the CNTS (controlled-not, not, Toffoli and swap) gate library must be even, since each of these gates are 3×3 or smaller. The authors of [40] prove that the converse of this is also true.

Definition 3.10 *Given an integer i, N^i denotes the circuit formed by placing an N gate on every wire corresponding to a 1 in the binary expansion of i. N^i may also refer to the permutation which this circuit computes.*

An example of a N^3 circuit is shown in Figure 5. All N^i circuits have the property that there can only be one not gate per wire. The authors prove that there are 2^n N-constructible permutations in S_{2^n}, and that the gate-minimal circuit for such a permutation is N^i for some i. The authors then extend these ideas from N-constructible permutations to T-constructible permutations.

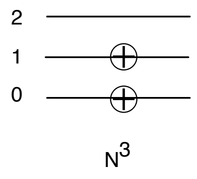


Figure 5: The circuit formed by placing a not gate on wires 0 and 1, referred to as N^3.

A particularly interesting idea next discussed by Shende *et al.* is that of circuit equivalences, based on concepts introduced by Iwama *et al.* [15]. The basic idea is related to Maslov *et al.*'s template technique, where possibly long circuits are simplified by replacing non-optimal sequences of gate with shorter, equivalent sequences. Shende *et al.*'s work proves a more general version of the work by Iwama *et al.*.

After a number of theorems and proofs on the constructibility of various types of reversible circuits, Shende *et al.* address the problem of finding optimal realizations for permutations known to be CNT-constructible.

A dynamic programming technique is used to build a library of small optimal circuits, followed by a depth-first search with iterative deepening technique to build the circuit. Memory is clearly an issue (for storage of the library), and despite the theoretical results, the authors themselves note that functions requiring nine or more gates tend to take more than 1.5 hours to synthesize, partly due to memory limitations.

3.4 A Canonical Form

As mentioned above, Iwama *et al.* introduce a set of transformations based on circuit equivalences in [15]. They also introduce a canonical form for reversible circuits.

Definition 3.11 *A quantum Boolean circuit S is said to be* of the canonical form *if (1) it has only CNOT gates whose target is the work bit, (2) it does not include two or more same CNOT gates, (3) CNOT gates are ordered lexiographically in terms of the indices of their control bits, and (4) no CNOT gates have auxillary inputs in their control bits.*

Figure 6 gives an example of two equivalent circuits, only one of which meets the given requirements for being canonical. The authors next make use of the positive polarity Reed-Muller format, defined as a Boolean

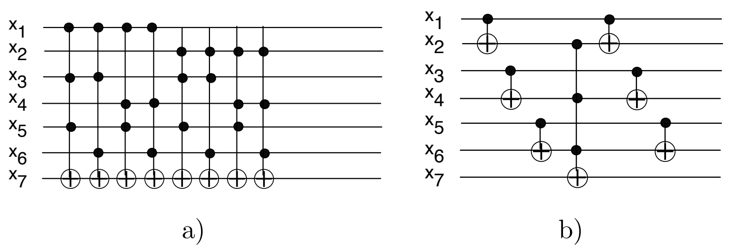


Figure 6: Two equivalent circuits, of which only a) is canonical [15].

formula of the form $a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus \cdots \oplus a_nx_n \oplus a_{1,2}x_1x_2 \oplus a_{1,3}x_1x_3 \oplus \cdots \oplus a_{n-1,n}x_{n-1}x_n \oplus \cdots \oplus a_{1,2,\ldots,n-1,n}x_1x_2\cdots x_{n-1}x_n$ where no negated literals are permitted and each a_i is a 0 or 1. It is known that any Boolean function can be expressed by a positive polarity Reed-Muller expression which is unique except for the order of terms, and the authors extend this concept to reversible expressions:

Lemma 3.3 *Any CNOT-based quantum circuit S has its unique canoncial form (as described in Definition 3.11).*

The proof involves using a positive polarity Reed-Muller expression of the function.

3.5 Regularity and Reversibility

A number of works including [17, 28, 27, 29] and [19] have suggested that regular reversible logic circuits are useful structures. These circuits are, as the name suggests, reversible, but also consist of identical reversible gates each of which is uniformly connected to neighbours in a regular fashion. In particular [28] suggests a lattice or net structure based on the Kerntopf [17] and Feynman gates. Such a structure, according to the authors, can realize an arbitrary symmetric function. They introduce a number of related theorems, which are reproduced here. First of all, the authors assume a sum-of-products type of representation of the functions to be implemented. These functions may or may not be reversible before beginning implementation; the implementation must, however, be reversible. In this context some definitions are presented.

Definition 3.12 *A variable appearing negated (non-negated) through-out a sum-of-products expression is called a* negative (positive) polarity variable.

Definition 3.13 *A* unate *function is a function expressed using only AND and OR operators (*e.g. *a sum-of-products) in which every variable has either positive polarity or negative polarity but not both.*

Definition 3.14 *A totally symmetric function having value 1 when exactly k of its n inputs are equal to 1 and value 0 otherwise is called a* single-index symmetric function *and is denoted by $S^k(x_1, x_2, \ldots, x_n)$. Similarly, $S\{i, j, k\}$ denotes the function that is 1 when exactly i, j, or k of its input variables are equal to 1.*

Secondly, the theorems below refer to MAX/MIN gates, which of themselves are not necessarily reversible. These are, in general, the multiple-valued equivalent of the traditional OR/AND gates (respectively). The MAX/MIN gate may be constructed out of two multiple-valued Fredkin gates, as detailed in [29].

Theorem 3.1 *Every positive unate symmetric function of n variables can be realized using $1+2+\cdots+n-1 = n(n-1)/2$ MAX/MIN gates.*

Theorem 3.2 *Every single index totally symmetric function of n variables can be realized using $n(n-1)/2$ MAX/MIN gates, $n-2$ fan-out gates and $n-1$ Feynman gates.*

Theorem 3.3 *Ever single-output totally symmetric function of n variables can be realized using $n(n-1)/2$ MAX/MIN gates, $n-2$ fan-out gates, $n-1$ Feynman gates in a second plane, and at most $n-1$ Feynman gates in a third plane.*

Theorem 3.4 *Every m-output totally symmetric function of n variables can be realized in $n(n-1)/2$ MAX/MIN gates, $n-1$ fan-out gates, $n-1$ Feynman gates in a second plane and at most $m(n-1)$ gates in the third plane.*

The authors are suggesting a stucture similar to that of a traditional programmable logic array (PLA), which consists of a plane of AND operators, the results of which can be interconnected through the use of a second plane consisting of OR operators. This structure, for instance, is very useful in implementing sum-of-product expressions. Perkowski *et al.* are suggesting three planes: one consisting of a regular structure that can implement all positive unate symmetric functions of its input variables; a second plane consisting of Feynman gates that converts the results of the first plane to single-index symmetric functions, and a third plane, also of Feynman gates, that can generate every output function as an EXOR of single-index symmetric functions from the second plane. This is possible through the application of the following equations, which are true for symmetric functions $S^{i1}(x_l, x_2, \ldots, x_n)$ and $S^{i2}(x_1, x_2, \ldots, x_n)$:

$$S^{i1} \text{ AND } S^{i2} = S^{i1 \cap i2} \tag{1}$$

$$S^{i1} \text{ OR } S^{i2} = S^{i1 \cup i2} \tag{2}$$

$$S^{i1} \text{ EXOR } S^{i2} = S^{i1 \text{ symm. diff. } i2} \tag{3}$$

$$S^{i} \text{ EXOR } S^{j} = S^{i} \text{ OR } S_{j} = S^{i,j} \tag{4}$$

3.6 A Technique using Reed-Muller Expansions

Agrawal and Jha [2, 12] proposed a technique that, according to them, produces near-optimal results and is the "first practical synthesis algorithm and tool for reversible functions with a large number of gates."

It is worth noting, however, that their first publication reports only results on reversible functions with three input variables although their second goes on to report results for four and five input variables. This technique uses a positive polarity Reed-Muller (PPRM) expansion (see Section 3.4) of all the outputs of the reversible function f to be synthesized; this expression is stored in a priority queue. The algorithm searches a tree of factors to determine the best possible solution. These authors are the only ones (to my knowledge) who are attempting to incorporate don't care values into their techniques.

3.7 ESOP-based Synthesis

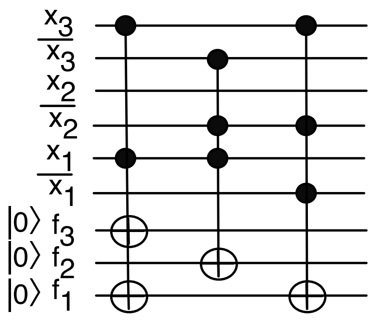
In recent discussions with M. Thornton he suggested an attractive synthesis technique based on exclusive-or sum-of-product (ESOP) representations. This technique is attractive for two reasons:

1. it requires as input a non-exponential representation of the funciton to be synthesized, and
2. it is so simple that actually performing the synthesis should prove to be very fast and potentially scalable to larger functions.

The problem with the technique, at least in its current unoptimized form, is that for a reversible circuit with n inputs it requires $3n$ qubits: one qubit for each input and one for the negated form of each input, and one qubit for each output. The technique consists of creating a circuit with the required number of qubits, and inserting one gate for each cube in the ESOP cubelist. If we allow CNOT (Toffoli) gates to control more than one output, then such a technique will result in a cascade with exactly p gates, p being the number of cubes (or products) in the ESOP cubelist. An example is shown in Figure 7. Work in implementing the

```
.i 3
.o 3
.p 3
1-1 101
001 010
100 001
.e
```

a)



b)

Figure 7: a) An ESOP cubelist. b) The resulting cascade of gates when generating a circuit from the cubelist in a).

basic technique is underway, and a paper introducing this technique is in preparation [8].

An additional advantage of this technique is that just as the cubes in the ESOP cubelist can be reordered without affecting the functions, the gates in the cascade can also be reordered with no impact on the implemented functions. This is not generally the case with reversible cascades, and suggests that optimizations could be introduced to advantage of this fact. We are particularly interested in reducing the number of qubits, and so one optimization suggested in discussions with Dr. Thornton consisted of sorting the cubes such that all cubes containing the non-negated form of a particular variable were moved to the top of the list, while all cubes contining the negated form of that same variable were moved to the bottom of the list. This process would then be repeated within each of the two halves of the list for the next variable, and so on. This would allow us to remove the qubits representing the negated form of each input variable and instead use NOT gates to switch as needed from the regular to negated "state". In the very worst case, where all minterms are included in the cubelist, this would result in $2^n - 1$ inverters, since the 2^n cubes could be ordered in such a way that only one bit changed between each cube. This is extremely unlikely, however, and so a cost function should be applied to determine which variables should be sorted on first – for instance, variables that do not use their negated version can appear anywhere in the ordering, while variables that are used extensively AND with a relatively even number of negated and non-negated appearances would benefit from being dealt with early on in the ordering process. In determining such a cost function both of these factors should be considered. Let us assign a value of $+1$ to each place where the variable appears as a 1 in the cubelist; a value of -1 to each place where the variable appears as a 0 in the cubelist, and a value of 0 if the variable appears as a don't care. Then we wish to maximize $\sum |x_i|$ to determine if a variable appears as a non-don't care value in a majority of the cubes, and to minimize $|\sum x_i|$ to determine if a variable appears fairly often in both its negated and non-negated forms, rather than just in one or the other.

$$C_{x_i} = p_1\left(\frac{1}{\sum |x_i|}\right) + p_2\left(\left|\sum x_i\right|\right) \tag{5}$$

Equation 5 gives a formula with two parameters that can be varied to alter the contribution of each part of the function, and each of the summations will run from cube 1 to cube p in the list. Experimentation has shown that setting the two parameters $p_1 = p_2 = 1$ allows the second portion of the equation, the $|\sum x_i|$ to overwhelm the first portion, and so the parameter p_1 should be set to some value to allow the two contributions to be more equal. It is likely that p_1 should vary with p (the number of cubes).

3.8 Decision Diagrams

A number of researchers have proposed the use of decision diagrams and their variants in reversible logic synthesis: Perkowski *et al.* [27], Kerntopf [18], Viamontes *et al.* [44, 45], and Thornton *et al.* [41]. We assume a basic knowledge of decision diagrams in this report.

3.8.1 Perkowski *et al.* [27]

Perkowski *et al.* suggest that in order to perform the processing described in [27],

> large multi-output functions should first be partitioned to smaller functions. This is based on their representations such as Binary Decision Diagrams, Pseudo-Kronecker Decision Diagrams (PKDD), Pseudo-Kronecker Diagrams with Complemented Edges (PDDDCE), Linearly Transformed Binary Decision Diagrams (LTBDDs), Function-drived Decision Diagrams (fDDs) or other similar diagrams.

They state also that "a fast natural mapping from a PKDD or PKDDCE diagram to a reversible netlist with Toffoli gates exists." The representation of a function as some sort of DD, with a subsequent mapping to an initial reversible implementation is considered to be a pre-processing step in this work with further transformations being applied to improve the final solution. The paper shows an example of mapping from a PKDDE and from a fDD. As the authors point out, although this technical report characterizes the technique as a DD-based method, the underlying concept of DDs is that the function is decomposed into various parts, be it using a Shannon decomposition, Ashenhurst-Curtis decomposition, or the new MP (multi-purpose Portland) decomposition as introduced by the authors.

3.8.2 Kerntopf [18]

Kerntopf uses a standard (RO)BDD to represent the reversible function, and each time a gate is added, transforms each of the output functions according to the effect of the gate, much like the process used by Maslov *et al.* in the template-based technique. The major difference is that Kerntopf stores the transformed function as a BDD rather than as a truth table. Kerntopf also introduces a new complexity measure:

Definition 3.15 *The* complexity measure $D(f)$ *of an* $n \times n$ *reversible function* f *is equal to* $D(f) = s(f) - n$ *where* $s(f)$ *denotes the number of non-terminal nodes in the ROBDD of* f *(assuming that complemented edges are used). Thus* $D(\text{identity function}) = 0$.

As gates are added and the functions are transformed, $D(f)$ should be strictly decreasing, until the final ROBDD has $D(f) = 0$. In every step of the algorithm *all* gates (Kerntopf uses the Not, Controlled-Not, Toffoli, Fredkin gate library) are examined and for each of them a SBDD for F^T is constructed. Gates are selected according to which generates a minimal transformed BDD, and if there are two or more gates with identical minimal BDDs the algorithm stores and processes all of them. Again, the results reported are limited to functions of size 3×3, but as the author notes, this technique has the potential to scale to larger functions due to its use of DDs rather than truth tables or PPRM representations. We will also point out that the author states that is he is studying modifications of the complexity measure, in order to incorporate more data and improve the algorithm's efficiency.

3.8.3 Viamontes *et al.* [44, 45]

Viamontes *et al.* [44] were the first to publish results and techniques for using DDs with reversible circuits. Their work reports results and alogorithms for a Quantum Information Decision Diagram (QuIDD) data structure, but their application is designed for simulation rather than synthesis, thus it will not be futher discussed in this report.

3.8.4 Thornton *et al.* [41]

Most recently Thornton *et al.* [41] have suggested a new decision diagram structure, with applications to quantum and reversible circuit simulation. Again, the stated purpose is primarily simulation, but the authors are investigating its use in synthesis as well. The underlying idea of QMDD (quantum multiple-valued decision diagrams) is that reversible and quantum gates (and cascades of such gates) can be represented by a matrix of size $2^n \times 2^n$ for an $n \times n$ reversible circuit/function/gate. For example, the functionality of the 3×3 Toffoli gate can be represented as a truth table, as shown in Table 9, or by the matrix shown in Figure 8. The rows and columns As shown in Figure 8 there is only ever one 1 in each row, and similarly in

$x\ y\ z$	$x\ y\ xy \oplus z$
000	000
001	001
010	101
011	011
100	100
101	101
110	111
111	110

Table 9: The truth table for the Toffoli gate that has x and y and control bits and z as the "work" or target bit.

each column. The 1's indicate a correlation between an input row in the truth table and an output, and so in Figure 8 it is evident that each set of input values from 000 to 101 results in the same set of values in the outputs, while the input values 110 have a 1 in the column corresponding with outputs 111, indicating the permuation that is taking place. Such matrices can then be represented by decision diagrams, as discussed in [4] and [11]. The authors suggest recursively partitioning the matrices into four blocks, the goal of which is to identify repeated blocks which can be shared. Each partitioning is identified by a vertex in a decision diagram, facilitating the sharing of identical blocks.

$$\begin{array}{c|cccccccc|}
000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
001 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
011 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
100 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
101 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
111 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111
\end{array}$$

Figure 8: The matrix representing the Toffoli gate whose truth table is given in Table 9. The binary values on the left and the bottom are included for clarity; those along the left represent the inputs to the truth table, while those along the bottom represent the outputs.

3.9 Spectral Techniques

Miller and Dueck [25, 24] have suggested the use of spectral techniques in reversible logic synthesis. Like Kerntopf's work, the authors suggest the use of a complexity measure. It is based on a Hamming count (the number of adjacent 0's and adjacent 1's in its Karnaugh map representation). This value can be computed by the following equation [14]:

$$C(f) = \frac{1}{2}\left(n2^n - \frac{1}{2^{n-2}}\sum_{v=0}^{2^n-1} ||v||r_v^2\right) \tag{6}$$

where $||v||$ is the number of 1's in the binary expansion of v and r_v is the v^{th} spectral coefficient resulting from an application of the Rademacher-Walsh transform. The reader is directed to [14] for details of spectral transforms and their uses in digital logic.

Miller and Dueck suggest a modified version of this complexity measure, combining the number of zero coefficients in the vector of spectral coefficients R and $C(f)$ as defined above. The measure is defined as:

$$D(f) = n2^n NZ(R) + C(f) \tag{7}$$

where $NZ(R)$ is the number of zero coefficients in R. The measure gives a higher value for functions with greater adjacency counts and for functions with more zero coefficients. Single variable functions, *e.g.* $f = x_0$ tend to have high adjacency counts and more zero coefficients in their spectral coefficients, while XOR functions also have more zero coefficients but lower adjacency counts.

The synthesis process is as follows:

- try putting a Fenman gate in all of the possible $n(n-1)$ positions; choose the position that results in the maximum positive change in $D(f_i)$.
- if there are no good choices resulting from the previous step try a Toffoli gate with two control lines in each of the $n(n-1)(n-2)$ possible positions each with the four possible negation patterns for the control variables; again choose the combination (if one exists) that results in the maximum positive change in $D(f_i)$.
- if there are no good choices resulting from the previous step try a Toffoli gate with three control lines according to the previous variations.
- if there are still no good choices the process ends in an error, otherwise $R'_j = R_j, j \neq i$ where x_i is the variable affected by the selected gate. R'_i is computed based on the gate.

Enhancements to this work were added in [25]. It is worth noting that the representation used in the examples is a reduced form of truth table; in this work a truth table such as the one shown in Table 10 is represented as a list of numbers giving the ordering of the output rows: $[0, 1, 2, 3, 4, 6, 5, 7]$.

xyx	$x'y'z'$
000	000
001	001
010	010
011	011
100	100
101	110
110	101
111	111

Table 10: A truth table showing the behaviour of the Fredkin gate with x as the control bit and y and z the "work" bits.

4 Previous work in Reversible Classification

Much of the previous work relating to classification of reversible functions has been in conjunction with the development of synthesis techniques. This section will repeat, briefly, some uses of classification that have been mentioned in Section 3 and add to these uses some additional techniques from the literature.

In Maslov *et al.*'s synthesis technique [22] the templates that are used in the simplification step have been classified to reduce the number of templates. The classes identified in this work were found by hand and verified by exhaustive enumeration. The templates are grouped together according to the number of gates within each, and according to their functionality, *e.g.* one class is referred to as the *passing rules* class. Various other criteria also are applied, such as the control and target lines within the template class being disjoint. However the important thing to remember is that these templates all implement the identify function; thus their use in simplification of an already synthesized function.

Van Rentergem *et al.* [31] classify various reversible gates as shown in Table 6. These classes are based on the behaviour of the gate, the number of control vs target lines, and the type of control function.

Although Iwama *et al.* [15] do not explicitly suggest a classification scheme, they do suggest a canonical form for the representation of reversible circuits. This suggests that their representation can be used in some classification technique, as transforming all circuits to their canonical representation would therefore group circuits with identical canonical forms in the same class.

Kerntopf [17] suggests that exhaustive study of 3×3 gates has led researchers to find classes of reversible 3×3 gates having the best compositional properties with respect to two-gate circuits. However this is clearly a very limited result. They did find that reversible logic circuits with regular structure tend to allow efficient realizations of Boolean functions when using reversible gates realizing simultaniously two-argument AND and OR operators at two of the outputs.

Additionally, Perkowski *et al.* [28, 29] have shown that regular reversible structures can realize arbitrary functions that possess unateness and symmetry properties, and in [27] they make mention of NPN classifications in relation to reversible logic synthesis, but do not clarify or elaborate on this.

In addition to the above, one special class of reversible circuits/functions is well known; the class of conservative circuits. According to [37], a logic element is called *conservative* if the weight of its input vector is equal, in all cases, to the weight of its output vector. Sasao and Kinoshita report some results on (mostly small) conservative logic circuits in [39, 37] and [38].

5 Possible Approaches

In this work I am not interested in classifying gates, or sub-circuits; rather the focus is on classification techniques for entire functions/circuits that may identify some technique that can be used in simplifying synthesis of reversible equivalents. As such, several possible approaches suggest themselves:

1. Classification of non-reversible Boolean functions to see which can be realized most easily using reversible gates. Of note would be unate symmetric functions, which can be implemented using the techniques suggested in [29].
2. Are there classes of functions which are easier to synthesize? For instance, using Van Rentergem's techniques, which type/class of arbitrary gate is easier to decompose [30]?
3. If one is using DDs to store transformed functions, then it would be useful to know which reversible functions are likely to have exponential sized DDs, and how often these are likely to be reached (using, for instance, Kerntopf's technique [18]).
4. Formal definitions of unateness, duality, and symmetry for reversible (and hence multi-output) functions would be useful to examine, and could lead to synthesis techniques for these special classes.

5.1 Symmetries, Duality and Unateness for Reversible Functions

Since the statement of these properties for reversible functions is simply a matter of extending the single-output definitions to multiple-output definitions, we will begin with this task. We also investigate how the NPN and spectral classifications may be extended to reversible functions, what the meaning of such extensions may be.

5.1.1 Symmetries

Definition 5.1 *A function $f(X)$ is* totally symmetric *if it is unchanged by any permutation of its n input variables [14].*

For instance, the function in Table 11 is a totally symmetric function. In general, a totally symmetric function can be constructed by assigning the same values to outputs where the input row has i 1's, as shown in Table 12. The implications of Table 12 are this: it is not possible to construct a totally symmetric

$x_2x_1x_0$	$f(X)$
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Table 11: The truth table for the majority function, a well-known totally symmetric (but not reversible) function.

reversible function for $n > 1$ since it requires that we violate the conditions that make a function reversible (*i.e.* it cannot be made bijective). In fact a relaxed form of symmetry, where permutation of some subset of

$x_2x_1x_0$	$f(X)$
000	a
001	b
010	b
011	c
100	b
101	c
110	c
111	d

Table 12: The only possible assignment for output-values of a function constructed to be totally symmetric.

the variables results in no change to the function outputs is also not permitted in a reversible function, for the same reason.

In identifying symmetries in reversible functions we must allow for the fact that every output of a reversible function must be unique, and thus any definition that requires that the output of the function be unchanged for some change to the inputs cannot be applied. There is one type of symmetry that may, however be applied; these symmetries have been discussed by a variety of authors under a variety of names such as antisymmetries [34, 33, 35] complement single variable symmetries [16], and skew symmetries [43]. We will refer to our own work and use our own naming convention in this report. Antisymmetries are a generalization of the symmetries of degree two, described in [14]. In general, a symmetry of degree two is identified when, for any two assignments to two input values, the function is unchanged. For instance, a function possesses an *equivalence symmetry* in $\{x_i, x_j\}$ if

$$f(x_1, \ldots, 0, \ldots, 0, \ldots, x_n) = f(x_1, \ldots, 1, \ldots, 1, \ldots, x_n)$$

and in general a symmetry of degree two exists when

$$f(x_1, \ldots, a, \ldots, b, \ldots, x_n) = f(x_1, \ldots, c, \ldots, d, \ldots, x_n)$$

for some $a, b, c, d \in \{0, 1\}$. An antisymmetry exists when

$$f(x_1, \ldots, a, \ldots, b, \ldots, x_n) = \overline{f}(x_1, \ldots, c, \ldots, d, \ldots, x_n)$$

Rice and Muzio [34] generalize the notation to be more meaningful in a multiple-valued logic setting: a partial symmetry, or symmetry of degree two is denoted as

$$P_{i,j}\{x_w, x_v\}$$

where i and j indicate the values assumed by x_wx_v, $i \neq j$ and $w \neq v$. An antisymmetry is then denoted as

$$\overline{P}_{i,j}\{x_w, x_v\}.$$

For example, the reversible function in Table 13 possesses the antisymmetry $\overline{P}_{0,3}\{y, z\}$.

xyz	$x'y'z'$
000	000 $\leftarrow$
001	001
010	010
011	111 $\leftarrow$
100	100 $\Leftarrow$
101	101
110	110
111	011 $\Leftarrow$

Table 13: A reversible function with the antisymmetry $\overline{P}_{0,3}\{y, z\}$, since $f(000) = \overline{f}(011)$ and $f(100) = \overline{f}(111)$.

How might this be useful in synthesis? If such symmetries can be detected, then it gives an insight into one possible route to take in a synthesis process.

Traditional definitions of symmetries concentrate on finding outputs rows in the truth table that are identical (or identical after negation) to each other. This is useful in traditional logic synthesis since fanout allows us to share outputs after one set of processing is performed. In reversible logic synthesis, though, traditional symmetries may not be as useful, partly due to the ban on fan-out. One possibility that may be useful, however, is identifying similarities between portions of the input side of the truth table and the output side of the truth table. For instance, would be very useful to know that only one variable is ever modified, In general most reversible gates pass a certain number of inputs unchanged through to the outputs. This, then, would be a useful thing to know about a reversible function – how many of the inputs arrive unmodified at the outputs.

5.1.2 Duality

Definition 5.2 *The* dual *of a function $f(X)$ is defined as $\overline{f}(\overline{X})$ where the bar within the brackets indicates that each of the function's variables x_i are individually complemented, and the bar over the f indicates that the output is also negated. It is denoted as $f^D(X)$ [14].*

Definition 5.3 *If $f^D(X) = f(X)$ then the function is said to be* self dual.

For example, the majority function $f(X) = x_1x_2 + x_2x_3 + x_1x_3$ is self-dual, as we can see in the truth table in Table 14. Table 15 shows how we might construct a self-dual function. We can assign arbitrary values

$x_3x_2x_1$	$f(X)$	$f(\overline{X})$	$\overline{f}(\overline{X})$
000	0	1	0
001	0	1	0
010	0	1	0
011	1	0	1
100	0	1	0
101	1	0	1
110	1	0	1
111	1	0	1

Table 14: An example of a self-dual single-output (non-reversible) function.

to the first half of the truth table, but for the bottom half the values must be the negation of their negated counterparts in the top half. Table 15 does not show any contradiction of the requirements for a reversible

$x_3x_2x_1$	$f(X)$
000	a
001	b
010	c
011	d
100	$\overline{d}$
101	$\overline{c}$
110	$\overline{b}$
111	$\overline{a}$

Table 15: Truth table showing how a $n = 3$ self-dual function might be constructed.

function, so we must conclude that it is possible to construct a reversible function that is also self-dual.

```
.i 4
.o 4
0000    1010
1111    1111
.e
```

Table 16: An example of a pla-style representation of a 4-input 4-output Boolean function.

The questions to be answered next are how to detect this property, and then how to use the information that it provides us with. For the latter question, this information certainly provides us with a decomposition that might be utilized.

5.1.3 Unateness

Definition 5.4 *A* unate *function is one in which no x_i input variable appears in both complemented and complemented form is a minimized sum-of-products expression for $f(X)$ [14].*

For example, $f(X) = x_1 + x_2 + x_3$ is a unate function, while $f(X) = x_1x_2 + \overline{x}_2x_3$ is not. As detailed in [28], unate functions can be nicely constructed using a grid structure that provides one version (*i.e.* negated or non-negated) of each variable to be used in building the function.

Since the definition of unateness depends on the use of a representation that is limited to a single-output, it is unclear how this can be extended to a multiple-output function such as a reversible function, or if there is any use in doing so. It is worth examining the pla-style of representation, such as shown in Table 16 to determine if it could be used in extending the concept of unateness to multiple-output functions. In Table 16 the first two lines indicate the number of inputs and outputs for the function, and the remaining entries (up to the .e flag signalling the end of the file) specify the relevant input assignments and their corresponding output assignments. It is generally assumed that any input assignments not listed in the file result in all 0's in the outputs. For use with general (non-reversible) Boolean functions a tool such as espresso [36] can be used to minimize the number of entries required, substituting explicit input assignments with don't cares (denoted by a -) where possible. In practice the file represents a two plane Programmable Logic Array (pla) where the first plane computes the AND (or product) of each variable, according to the values given in each row, and the second plane computes the OR of each row, or products resulting from the first plane's computations.

Future work will examine how practical this representation is for reversible logic functions (*i.e.* how much reduction through minimization can be achieved for a general reversible function) and whether it can be used to extend the concept of unateness.

5.1.4 NPN and Spectral Classifications

NPN and spectral classification techniques were briefly mentioned in Section 2.3. In general, these are applied to single-output functions, but extensions of spectral techniques to multiple-valued functions have been fairly well researched (see for instance, [23]). Extensions to multiple-output functions are uncommon, with most researchers choosing to compute spectral coefficient vectors for each individual function and manage them separately, as in [25]. It is worth detailing both the NPN and spectral techniques as applied to single-output functions in order to gain some understanding of how they may be extended to reversible functions.

The NPN classes are classes of Boolean function in which all functions in the class can be "reached" from any other function in the class by applying one or more of the NPN operations. The NPN operations are as follows:

xyz	$x'y'z'$	$x'y'z'$
000	001	1
001	010	2
010	011	3
011	100	4
100	101	5
101	110	6
110	111	7
111	000	0

Table 17: An example of converting a reversible function that computes a mod 8 increment from three outputs to a single multiple-valued output.

- negation of a variable,
- permutation of variables, and
- negation of the output.

For instance, the function $f(X) = x_1x_2 + x_3$ is in the same NPN class as the function $f(X) = \overline{x}_1x_2 + x_3$ (obtained by negating x_1), and also in the same NPN class as $f(X) = \overline{x}_3x_2 + x_1$ (obtained by permuting x_1 and x_3) and $f(X) = (x_3 + \overline{x}_2)\overline{x}_1$ (obtained by negating the output of the function). The interesting part about these invariant operations is that each of them can be implemented quite nicely with a reversible gate: negation is itself reversible, and permutation can be implemented with a swap gate. The only problem is that this classification technique is well-known and well-researched for single-output functions, but not for multiple-output functions such as reversible functions. One way to extend it could be to treat the outputs as a single, multiple-valued variable, as show in Table 17. The effect of the NPN operations can be described in terms of the truth table as follows:

- negation of an input variable x_i: swap each row

  $$x_n, \ldots, x_{i+1}, 1, x_{i-1}, \ldots, x_1$$

  with the corresponding row

  $$x_n, \ldots, x_{i+1}, 0, x_{i-1}, \ldots, x_1.$$

  In Table 17 if we negate variable z output values 1 and 2 would be swapped, as would be 3 and 4, 5 and 6, and 7 and 0.

- permutation of two input variables $x_i, x_j (i \neq j)$: rows where $x_i = x_j$ are unaffected, while each row

  $$x_n, \ldots, x_{j+1}, a, x_{j-1}, \ldots, x_{i+1}, b, x_{i-1}, \ldots, x_1$$

  $a, b \in \{0, 1\}, a \neq b$ is swapped with the corresponding row

  $$x_n, \ldots, x_{j+1}, b, x_{j-1}, \ldots, x_{i+1}, a, x_{i-1}, \ldots, x_1.$$

  For example if we permute variables y and z in Table 17 we swap rows 2 and 3, and rows 5 and 6.

- negation of the output: this operation results in each output row being negated; in Table 17 row 0 is swapped with row 7, row 1 with row 6, row 2 with row 5, and row 3 with row 4.

This author sees no reason why the NPN classes cannot be extended in this way to reversible functions, and why this knowledge cannot be used in creating a synthesis process for reversible functions. For example, given the function shown in Table 18a) we can apply a NOT gate to negate x_1, resulting in the truth in Table 18b), then permute x_2 and x_3 using a Swap gate, resulting in the identity function as shown in Table 18c). Figure 9 gives the cascade of gates that could be used to implement this function. Of course, the example shown in Table 18 was constructed by taking the identify function and applying NPN operations on it; not all functions will lie in the same class as the identity function. Thus in order to use this technique there are two requirements:

$x_3x_2x_1$	$x'_3x'_2x'_1$
000	001
001	000
010	101
011	100
100	011
101	010
110	111
111	110

a)

$x_3x_2\overline{x}_1$	$x'_3x'_2x'_1$
000	000
001	001
010	100
011	101
100	010
101	011
110	110
111	111

b)

$x_2x_3\overline{x}_1$	$x'_3x'_2x'_1$
000	000
001	001
010	010
011	011
100	100
101	101
110	110
111	111

c)

Table 18: An example of using NPN operations to convert a function to the identity.

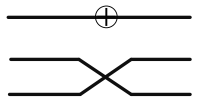


Figure 9: The cascade of gates resulting from the operations applied in Table 18.

- first, that we identify for each NPN class of reversible functions a unique and canonical function for which there is a known and optimal reversible implementation, and
- second that we can identify both the class to which a reversible function belongs and the operations necessary to transform it to the canonical representative.

Since this technique may be related to or improved upon by the use of spectral techniques, we will consider spectral classification for reversible functions before building any futher upon this idea.

Briefly, in order to transform a function from the Boolean domain to what is called the *spectral domain*, some type of function or transform is applied. The resulting coefficients are called the *spectral coefficients* of the function. The vector of spectral coefficients is referred to as R while the output vector of the function $f(X)$ is referred to as Z. The spectral transform is then computed as

$$R = T^n \times Z \tag{8}$$

One of the commonly used spectral transforms is the Hadamard transform. It is defined as [14]

$$\begin{gathered} T^n \triangleq \begin{bmatrix} T^{n-1} & T^{n-1} \\ T^{n-1} & -T^{n-1} \end{bmatrix} \\ \text{where} \\ T^0 \triangleq [1] \end{gathered} \tag{9}$$

An example of computing the spectral coefficients for a three-variable single-output Boolean function is shown in Figure 10. Other transformations that are commonly used are the Walsh and Rademacher-Walsh. Definitions of these transforms are given in [14]. The spectral coefficients generated by either the Walsh, Rademacher-Walsh, or the Hadamard transforms are the same, with the values appearing in different orderings. It should be noted that in Figure 10 the function output vector for a Boolean function takes on values from $\{0, 1\}$. The functions in the transform matrix, however are encoded as $\{+1, -1\}$. It is common to re-encode the function output vector in $\{+1, -1\}$. In this case the output vector is referred to as Y, and the resulting spectral coefficients are referred to as S. For each of the transforms discussed above, the resulting coefficients are labeled as shown in Figure 10. This labeling varies depending on the ordering of the rows in the transform matrix, and reflects the function comparison that is being performed with each row multiplication.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{array}{c} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{array} = \begin{bmatrix} 7 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{array}{c} r_0 \\ r_1 \\ r_2 \\ r_{12} \\ r_3 \\ r_{13} \\ r_{23} \\ r_{123} \end{array}$$

Figure 10: Computing the spectral coefficients using the Hadamard transform matrix.

5.2 PPRMs

This section discusses PPRM representations used to specify or manage reversible functions in some of the above described synthesis techniques. Specifically, we are asking two questions regarding the use of these representations:

1. Are there reversible functions for which there are "good" *i.e.* small PPRM representations, and similarly are there "bad" functions?
2. If a function is good/bad does this correspond to either a good/bad implementation, or to a short/long synthesis process?

Drechsler *et al.* [6] discusses OFDD based minimization of fixed polarity Reed-Muller expressions, and points out that

> ... AND/EXOR realizations are very efficient for large classes of circuits, *e.g.* arithmetic circuits, error correcting circuits, and circuits for tele-communication. For these classes EXOR-circuits, derived from Reed-Muller expressions (RMEs), need less gates for the representation of a Boolean function and drastically reduce the number of signals in the resulting network.

One can assume that similar results would hold for reversible functions since the works reported here represent each of the reversible function outputs as a separate PPRM expression. This implies that PPRM-based techniques might be better to choose for the class of circuits described above. Certainly DD-based techiques generally have the result that a smaller representation leads to a smaller circuit and also to faster synthesis. More work is needed to verify that this holds, for example, in the technique reported by Gupta *et al.* [12]. An interesting extension of Gupta's work would be to use Drechsler's OFDDs to store and manage the PPRMs in implementing Gupta's algorithm; since a large part of the algorithm involves the identification and substitution of factors it is likely that this process could be managed effectively and efficiently through the use of a DD structure.

6 Future Work

There are a number of interesting questions that are raised in this report. Some of these are given below:

- Since two-variable (anti-)symmetries can certainly exist in reversible logic functions, how can this be identified and leveraged in a synthesis process?

- Similarly, self-dual reversible logic functions are also possible, and can provide a first step in decomposing such a function into two smaller, and possibly easier to synthesize functions. How can we detect this property, and how would such a decomposition play a part in synthesis?
- There seems to be little work extending NPN classes to multiple-output functions, let alone to reversible logic functions. This is interesting, and begs the question as to why this is so. Certainly further investigation into the literature is warranted, and consideration as to how NPN classification, as used in traditional Boolean function logic synthesis, could be extended to reversible logic synthesis would also be an interesting avenue to pursue.
- One of the key problems in nearly all of the synthesis techniques surveyed in this paper is that of representation. Many of them rely on a truth-table or related style of representation, which is always going to be exponential in size. This suggests that more effort in comparing representation techniques, and the parts they play in synthesis, would be a useful direction to research.

7 Summary

This report provides a summary of the state of the current literature surrounding logic synthesis techniques for reversible logic functions. The purpose of this is to provide some background in the real direction of this report: that of investigating the use of classification as applied to reversible logic functions. Classification has been well researched for traditional Boolean logic functions, and in this area has been a useful tool. We argue that many of the concepts could be extended for reversible functions, and leveraged in some of the same ways.

References

[1] International Technology Roadmap for Semiconductors (ITRS). http://public.itrs.net, 2002 update.

[2] A. Agrawal and N. K. Jha. Synthesis of Reversible Logic. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1384–1385, 2004. 16–20 February, Paris, France, IEEE Computer Society.

[3] C. H. Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development*, 6:525–532, 1973.

[4] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral Transforms for Large Boolean Functions with Application to Technology Mapping. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, 1993.

[5] E. P. DeBenedictis. Reversible Logic for Supercomputing. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 391–402, 2005. May 4–6, Ischia, Italy, ACM Press.

[6] R. Drechsler, M. Theobald, and B. Becker. Fast OFDD based Minimization of Fixed Polarity Reed-Muller Expressions. In *Proceedings of the European Design Automation Conference (EURO-DAC94)*, pages 2–7, 1994.

[7] C. Edwards. The Application of the Rademacher-Walsh Transform to Boolean Function Classification and Threshold Logic Synthesis. *IEEE Trans. on Comp.*, C–24(1):48–62, January 1975.

[8] K. Fazel, M. Thornton, and J. E. Rice. ESOP-based Toffoli Gate Cascade Generation. In *to appear in the Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2007.

[9] E. Forsberg. The Electron Waveguide Y-Branch Switch: a Review and Arguments for its Use as a base for Reversible Logic. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 404–406, 2005. May 4–6, Ischia, Italy, ACM Press.

[10] M. P. Frank. Introduction to Reversible Computing: Motivation, Progress, and Challenges. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 385–390, 2005. May 4–6, Ischia, Italy, ACM Press.

[11] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Formal Methods in System Design*, 10(2–3):149–169, April 1997.

[12] P. Gupta, A. Agrawal, and N. K. Jha. An Algorithm for Synthesis of Reversible Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2317–2330, Nov. 2006.

[13] S. Hurst. *The Logical Processing of Digital Signals.* Crane Russak, 1978.

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

[15] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation Rules for Designing CNOT-based Quantum Circuits. In *Proceedings of the 39th Design Automation Conference*, pages 419–424, 2002.

[16] S. Kannurao and B. J. Falkowski. Identification of Complement Single Variable Symmetry in Boolean Functions Through Walsh Transform. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pages 745–748, 2002.

[17] P. Kerntopf. Synthesis of Multipurpose Reversible Logic Gates. In *Proceedings of the Euromicro Symposium on Digital System Design: Architectures, Methods and Tools (DSD)*, pages 269–267, 2002.

[18] P. Kerntopf. A New Heuristic Algorithm for Reversible Logic Synthesis. In *Proceedings of the Design Automation Conference (DAC)*, pages 834–837, 2004. June 7–11, San Diego, CA, USA, ACM.

[19] A. B. Khlopotine, M. Perkowski, and P. Kerntopf. Reversible Logic Synthesis by Iterative Compositions. In *Proceedings of the International Workshop on Logic Synthesis (IWLS)*, 2002.

[20] S. Kim and S.-I. Chae. Implementation of a Simple 8-bit Microprocessor with Reversible Energy Recovery Logic. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 421–426, 2005. May 4–6, Ischia, Italy, ACM Press.

[21] R. Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[22] D. Maslov, G. W. Dueck, and D. M. Miller. Synthesis of Fredkin-Toffoli Reversible Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(6):765–769, June 2005.

[23] D. M. Miller. Spectral Symmetry Tests. In *The Proceedings of the 11th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 130–134, 1981.

[24] D. M. Miller. Spectral and Two-Place Decomposition Techniques in Reversible Logic. In *Proceedings of the IEEE Midwest Symposium on Circuits and Systems (MWCAS)*, pages II493–II496, 2002.

[25] D. M. Miller and G. W. Dueck. Spectral Techniques for Reversible Logic Synthesis. In *Proceedings of the 6th International Symposium on Representations and Methodology of Future Computing Technologies*, 2002.

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

[27] M. Perkowski, L. Joziwak, A. Mixhchenko, A. Al-rabadi, A. Coppola, A. Buller, X. Song, M. Khan, S. Yanushkevich, V. P. Shmerko, and M. Chrzanowska-Jeske. A General Decomposition for Reversible Logic. In *Proceedings of the International Workshop on Methods and Representations (RM)*, pages 119–138, 2001. August 10–11, Starkville, Mississippi, USA, RM Workshop.

[28] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, and A. Coppola. Regularity and Symmetry as a Base for Efficient Realization of Reversible Logic Circuits. In *Proceedings of the International Workshop on Logic Synthesis (IWLS)*, 2001.

[29] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Jozwiak, A. Coppola, and B. Massey. Regular Realization of Symmetric Functions using Reversible Logic. In *Proceedings of the Euromicro Symposium on Digital System Design (DSD)*, pages 245–252, 2001.

[30] Y. Van Rentergem, A. De Vos, and K. De Keyser. Using Group Theory in Reversible Computing. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC2006)*, pages 2397–2404, 2006.

[31] Y. Van Rentergem, A. De Vos, and L. Storme. Implementing an Arbitrary Reversible Logic Gate. *Journal of Physics A: Mathematical and General*, 38(16):3555–3577, April 2005.

[32] J. E. Rice. An Introduction to Reversible Latches, 2007. submitted to The Computer Journal, March 2007.

[33] J. E. Rice and J. C. Muzio. Antisymmetries in the Realization of Boolean Functions. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 2002. CD ROM paper number 2666.

[34] J. E. Rice and J. C. Muzio. A Characterization of Antisymmetry in Boolean and Multi-Valued Functions. In *Proceedings of the International Symposium on Multiple-Valued Logic (ISMVL)*, pages 270–275, 2005.

[35] Jacqueline E. Rice. *Autocorrelation Coefficients in the Representation and Classification of Switching Functions*. PhD thesis, University of Victoria, 2003.

[36] R. Rudell. Espresso minimization tool man pages.

[37] T. Sasao and K. Kinoshita. Cascade Realization of 3-Input 3-Output Conservative Logic Circuits. *IEEE Transactions on Computers*, C-27(3):214–221, March 1978.

[38] T. Sasao and K. Kinoshita. Realization of Minimum Circuits with Two-Input Conservative Logic Elements. *IEEE Transactions on Computers*, C-27(8):749–752, Aug. 1978.

[39] T. Sasao and K. Kinoshita. Conservative Logic Elements and Their Universality. *IEEE Transactions on Computers*, C-28(9):682–685, Sept. 1979.

[40] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of Reversible Logic Circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, June 2003.

[41] M. Thornton, D. M. Miller, and D. Goodman. A Decision Diagram Package for Reversible and Quantum Circuit Simulation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 8597–8604, July 2006. held at the IEEE World Congress on Computational Intelligence (WCCI).

[42] T. Toffoli. Reversible Computing. Technical Report MIT/LCS/TM No. 151, MIT, 1980.

[43] C.-C. Tsai and M. Marek-Sadowska. Generalized Reed-Muller Forms as a Tool to Detect Symmetries. *IEEE Trans. on Comp.*, 45(1):33–40, January 1996.

[44] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based Simulation of Quantum Computation in the Density Matrix Representation. *Quantum Information & Computation*, 5(2):113–130, 2005. preprint available at quant-ph/0403114.

[45] G. F. Viamontes, I. L. Markov, and J. P. Hayes. QuIDDPro: High-Performance Quantum Circuit Simulation, 2006. http://vlsicad.eecs.umich.edu/Quantum/qp/.

[46] A. De Vos and Y. Van Rentergem. Reversible Computing: From Mathematical Group Theory to Electronical Circuit Experiment. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 35–44, 2005. May 4–6, Ischia, Italy, ACM Press.