New Considerations for Spectral Classification of Boolean Switching Functions

J.E. Rice, J.C. Muzio, and N. Anderson

January 2011

This article was originally published at:

http://dx.doi.org/10.1155/2011/356137

Hindawi Publishing Corporation
VLSI Design
Volume 2011, Article ID 356137, 9 pages
doi:10.1155/2011/356137

Research Article

New Considerations for Spectral Classification of Boolean Switching Functions

J. E. Rice,[1] J. C. Muzio,[2] and N. Anderson[2]

[1] *Department of Mathematics & Computer Science, University of Lethbridge, 4401 University Drive West, Lethbridge, AB, Canada T1K 3M4*
[2] *Department of Computer Science, University of Victoria, P.O. Box 3055 STN CSC, Victoria, BC, Canada V8W 3P6*

Correspondence should be addressed to J. E. Rice, j.rice@uleth.ca

Received 5 May 2010; Revised 21 October 2010; Accepted 11 January 2011

Academic Editor: Avi Ziv

This paper presents some new considerations for spectral techniques for classification of Boolean functions. These considerations incorporate discussions of the feasibility of extending this classification technique beyond $n = 5$. A new implementation is presented along with a basic analysis of the complexity of the problem. We also note a correction to results in this area that were reported in previous work.

1. Introduction

In building or designing a circuit or Boolean switching function, one approach is to begin with a known implementation and add logic to achieve the desired functionality. However, as n, the number of inputs to the switching function, increases, it is impossible to individually examine each switching function to determine if such a technique will lead to a reasonable implementation, both for a given starting function and for the desired function. Thus it is desirable to determine some method of grouping or classifying these functions, such that functions with similar characteristics fall into a common class. One such technique involves applying a transform to the outputs of the switching function, thus generating a series of coefficients which are then used to group the functions. This was first suggested by Edwards [1]. Since then, other researchers have expanded upon this in works such as [2, 3]. The limiting factor of such an approach has been the computational complexity of applying the transform.

For many spectral classification approaches, the transforms generally used are the Walsh or Rademacher-Walsh transforms. These transforms are desirable for a number of reasons, but techniques using these transforms are quite computationally intensive. They are so computationally intensive that it has not been possible to compute the complete spectral classes for functions with more than 5 variables, as published in [3, 4].

In this paper, we revisit the computation of the spectral classes. Since 30 years have passed since this problem has been examined, we theorized that advances in technology could allow for further classes to be computed. We found this to be untrue and present some basic analysis of the problem to support this. In our attempt to press beyond the $n = 5$ limitation, however, we developed a new approach to spectral classification, and during testing we found that our results differed from those presented in [3]. An additional contribution of our work is a correction to the results from [3], and an argument as to why our new results are correct.

The main idea of our approach involves applying the operations used in classification directly to the functions themselves, using bit operations whenever possible. Rules reflecting the changes for each operation are also predetermined where possible, and stored as templates in order to speed up overall processing. Unlike other approaches, where the spectral domain is used for processing, we chose to remain in the functional domain and make use of a truth table representation for each function, thus allowing us to use a very simple bit-vector as the underlying data structure. A variety of optimization techniques are used to speed up the processing; however, we avoid assumptions made by

previous researchers that resulted in faulty classifications when extended to larger values of n.

2. Background

We first provide some background in the areas of classification and in particular, spectral classification.

2.1. Classification. Given that 2^{2^n} unique Boolean switching functions can be realized for n input variables, it is clear that having some technique for grouping functions together according to some similarities would be useful. There are many techniques for classification, but in general a classification of a set of functions F into classes $Q_1, Q_2, \ldots, Q_p$ based on transformations $T_1, T_2, \ldots, T_m$ is such that

$$F = Q_1 \cup Q_2 \cup \cdots \cup Q_p, \qquad Q_i \cap Q_j = \varnothing. \tag{1}$$

Two functions f_i and f_j, $i \neq j$ are in the same class Q_k if and only if f_i can be obtained from f_j by the application of some appropriate set of transformations from $T_1, \ldots, T_m$. No set of transformations applied to a function in Q_i can lead to a function in Q_j for any $i, j \in \{l, \ldots, p\}$ where $i \neq j$.

2.2. Spectral Classification. One of the more common classification techniques divides functions into the NPN classes [4]. In this technique the transformations consist of negation of the input variable(s) (N), permutation of the input variable(s) (P), and negation of the output (N).

The negation of an input variable x_i effectively replaces each instance of a value with its inverse for that particular variable. An example is shown in Table 1. As this example shows, the effect of negating an input variable is that the rows of the truth table for the switching function are permuted.

Similarly, the permutation operation also affects the rows of a function's truth table. Negation of the output has the effect of inverting (negating) all the output values, and so the rows of the table are not affected.

The spectral classification technique is based on examination of a function's spectral coefficients. Computation of these coefficients is discussed in Section 2.3. There are five operations which may be applied to a function that do not change the values of a function's spectral coefficients, only the order in which they appear. These five operations are [3]

(1) permutation of the input variables,

(2) negation of the input variables,

(3) negation of the output,

(4) replacing some variable x_i with $x_i \oplus x_j$, $i, j \in \{1, \ldots, n\}$ and $i \neq j$, and

(5) replacing the output $f(X)$ with $f(X) \oplus x_i$ for some $i \in \{1, \ldots, n\}$.

As one can see, the first three operations are the same as given for the NPN classification. These five operations are sometimes referred to as invariance operations. Throughout this work we refer to each of these operations according to their numeric ordering; for example, a Type 1 invariance operation would refer to some permutation of the function's input variables.

Table 1: The effect of negation of an input variable (x_1) on the truth table of a Boolean switching function. Note that the output vector of the function is generally referred to as Z, with z_i being individual elements of Z.

$x_3x_2x_1$	$f(X)$		$x_3x_2\overline{x}_1$	$f(X)$
000	z_0		000	z_1
001	z_1		001	z_0
010	z_2		010	z_3
011	z_3	$\longrightarrow$	011	z_2
100	z_4		100	z_5
101	z_5		101	z_4
110	z_6		110	z_7
111	z_7		111	z_6

2.3. Computing the Spectral Classes. The spectral classes are formed by determining all functions that have the same set of absolute values of spectral coefficients. Functions with the same spectral coefficients, disregarding ordering and sign, fall into the same spectral class. It is generally necessary to compute the spectral coefficients for a function to determine its spectral classification.

Computation of the spectral coefficients can be carried out by applying the appropriate spectral transform to the output vector of the Boolean switching function in question. In this explanation we describe the use of the Hadamard transformation matrix to compute the spectral coefficients. It should be noted that this results in the same overall set of coefficients as do the Walsh and Rademacher-Walsh transforms; the primary difference is in the resulting ordering of the values. The Hadamard transformation matrix was chosen simply because of its recursive nature, making it easy to describe. Since the remainder of this paper focuses on computing the spectral classes in the functional domain and not in the spectral domain this explanation is included merely for the sake of completeness and understanding of the work.

The Hadamard transformation matrix is recursively defined as given in (2).

$$T^n = \begin{bmatrix} T^{n-1} & T^{n-1} \\ T^{n-1} & -T^{n-1} \end{bmatrix} \quad \text{where } T^0 = 1. \tag{2}$$

Then computation of the spectral coefficient vector S is generally computed by applying (3),

$$S = T^n Y, \tag{3}$$

where Y is the output vector of the switching function in question. Generally, $\{+1, -1\}$ encoding is used; that is, 0 values are replaced with $+1$ and 1 values are replaced with -1. The output vector is referred to as Y if $\{+1, -1\}$ encoding is used, and Z if $\{0, 1\}$ encoding is used. Discussions of this replacement can be found in [3, 5].

An example of applying the Hadamard transformation matrix to the $\{+1, -1\}$ encoded output vector of the function $f(X) = \overline{x}_3\overline{x}_1 + x_2x_1 + x_3\overline{x}_2$ is shown in Figure 1. A straightforward application of (3) would result in the summation of $2^n \times 2^n$ individual product terms; however, there are fast transform procedures that allow interim values to be reused, thus saving some computational effort. Details of such procedures are given in [3].

2.4. Relationship between Spectral Classes and the Spectral Coefficients. Before discussing the impact of the five invariance operations, it is useful to define some labeling to clarify our discussion. If we describe the spectral coefficients as a vector of values S consisting of individual coefficients s_α, then it is common to label each individual coefficient according to its meaning. For instance, a coefficient with a single subscript, for example, s_i, indicates the correlation between $f(X)$ and the input variable x_i. A coefficient with multiple subscripts, for instance s_{ij}, $i \neq j$ indicates the correlation between $f(X)$ and the function $x_i \oplus x_j$. The remaining coefficient, s_0 indicates the similarity of $f(X)$ to the constant function. An example illustrating this labeling is shown in Figure 1. Given this labeling we can then describe the effect of the five invariance operations on the spectral coefficients of a function as follows.

(1) Permutation of any input variables x_i and x_j results in the exchange of s_i with s_j, s_{ik} with s_{jk}, s_{ikl} with s_{jkl}, and so on. Coefficients s_0, s_k, s_{ij}, and others in this pattern remain unchanged.

(2) Negation of any input variable x_i results in the negation of the related coefficients: s_i, s_{ij}, s_{ik}, and so forth. Other coefficients s_0, s_j, s_{jk}, and so on remain unchanged.

(3) Negation of the output results in the negation of all 2^n spectral coefficients.

(4) Replacement of any variable x_i with $x_i \oplus x_j$ results in the exchange of s_i with s_{ij}, s_{ik} with s_{ijk}, s_{ikl} with s_{ijkl}, and so on. Coefficients $s_0, s_j, s_{jk}, \ldots$ remain unchanged.

(5) Replacement of the output $f(X)$ with $f(X) \oplus x_i$ results in the exchange of s_i with s_0, s_{ij} with s_j, s_{ijk} with s_{jk}, and so on. All coefficients are affected by this change [3].

For example, given $f(X) = \overline{x}_3\overline{x}_1 + x_2x_1 + x_3\overline{x}_2$, then functions $g(X) = x_3\overline{x}_1 + x_2x_1 + \overline{x}_3\overline{x}_2$ and $h(X) = \overline{x}_2\overline{x}_1 + x_3x_1 + x_2\overline{x}_3$ belong to the same spectral class. $g(X)$ can be generated by negating variable x_3 in the original function $f(X)$, and $h(X)$ can be generated by permuting variables x_2 and x_3 in the original function. $f(X)$ and $g(X)$ would have the same spectral coefficients, in the same order, while the change for $h(X)$ would be the interchange of coefficient pairs s_2 and s_3 and s_{12} and s_{13}.

Thus the spectral classes can be computed by determining all functions with the same sets of coefficient values, or alternatively by applying the 5 invariance operations to a function in order to generate all other functions in the same class.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} -4 \\ 0 \\ 0 \\ -4 \\ 0 \\ -4 \\ 4 \\ 0 \end{bmatrix} \begin{matrix} s_0 \\ s_1 \\ s_2 \\ s_{12} \\ s_3 \\ s_{13} \\ s_{23} \\ s_{123} \end{matrix}$$

FIGURE 1: An example of computing the spectral coefficients for the function $f(X) = \overline{x}_3\overline{x}_1 + x_2x_1 + x_3\overline{x}_2$.

3. Related Work

There has been a variety of approaches suggested for computation of the spectral coefficients. For instance, Thornton and Drechsler propose the computation of spectral information from logic netlists in [6], and also suggest techniques based on AND/OR graphs and output probabilities [7–9]. Clarke et al. [10] is probably one of the earliest researchers to suggest computation based on decision diagrams, and other researchers such as Jankovic et al. have followed up on this [11–15]. Techniques based on programmable hardware have also been introduced [16] as have parallel techniques [17].

One of the reasons for this renewed interest in spectral coefficients is that researchers such as Hansen and Sekine [18] have suggested new synthesis techniques based on the spectral coefficients of a function. Other work in this area includes techniques for Boolean matching [19], in reversible logic synthesis [20, 21], and in multiple-valued applications [22]. The reader is directed to [23] for further details on advancements in spectral techniques in this field as well as various works by Falkowski and Yan including [24, 25].

While there are many questions of interest in the area of classification, research here has not been as common. Classification techniques based on Reed-Muller forms have been suggested by Tsai and Marek-Sadowska in [26], and a technique based on the matrix representation of a function was introduced by Lapshin in [27]. Related work on classification has also been discussed by Rice and Muzio in [28]. Strazdins [29] discusses the matter of Boolean function classification from a more mathematical approach, although it is of interest to note that this work also is following up on similar work after an approximately 30-year gap.

Our research has focused primarily on spectral classification as introduced by Edwards. In [1] Edwards applied five spectral invariance operations to classify all 2^{2^5} possible Boolean switching functions for $n \leq 5$. He did so using a compressed version of the spectral coefficients referred to as a function's *signature*. This signature consists of s_0, all coefficients with a single subscript, and a list of the absolute values of all coefficients with the number of occurrences of each value. For example, the spectral coefficients for the function $f(X) = \overline{x}_3\overline{x}_1 + x_2x_1 + x_3\overline{x}_2$ are shown in Figure 1, while the corresponding signature for this function is $-4; 0; 0; 0; 4 \times 0; 4 \times 4$. Thus $f(X)$ has the coefficient values

$s_0 = -4$, $s_1 = 0$, $s_2 = 0$, and $s_3 = 0$, and the remaining coefficients are grouped into the summary indicated by the notation showing that there are four coefficients with the value 0 and four coefficients with the (absolute) value 4; note that the values for s_0, s_1, s_2, and s_3 are also included in this summary. As one can see, the signature compresses the entire coefficient information by removing the information relating to where in the coefficient pattern a particular value appears.

It was thought that this signature was sufficient to uniquely define each class, and Edwards stated that 47 spectral classes were required to completely classify all 2^{2^5} functions. However, further investigation in [4] discovered that one of the 47 classes published in [1] did not meet the definition of the classification. There existed at least one function within a particular class for which there was no way to transform it to any other function in that class. This offending class was therefore split into two separate classes, each with the same signature. This discovery proved that this spectral signature as defined by Edwards is not sufficient to uniquely classify all 2^{2^n} functions and the number of classes for $n \leq 5$ thus increased from 47 to 48.

A complementary approach was taken in [3] using only the first four operation types. This approach produced 191 classes, which were mapped to the equivalent classes in [4] using the signature of each class. Clearly the classes determined using only four of the invariance operations have fewer criteria than the classes determined using all five operations, and so there are multiple classes defined in [3] for each class defined in [4].

The general process used in [1, 3, 4] was to transform a starting function, f, into the spectral domain and attempt to construct all other functions in the same spectral class using the five spectral transformations. In order for this to be computed in a reasonable amount of time for $n = 5$, the problem was extensively pruned and optimized. Unfortunately, the details of these optimizations are unavailable.

4. Our Approach

In contrast with the previous work in [1, 3], the processing in our approach takes place entirely in the functional domain, with the exception of the direct comparison to the previous work. An advantage of performing classification in the spectral domain, or in other words, manipulating the spectral coefficients of a function, is that the spectral coefficients provide more global information about a function, making it possible to make observations about groups of functions and optimize accordingly. However, a big disadvantage of this approach is in the overhead of computing the spectral coefficients, and even fast techniques still have significant computational complexity.

To avoid the overhead associated with converting to the spectral domain, we chose to focus on the functional domain and apply the invariance operations directly to the function's truth table representation. Since most of the operations simply require reordering of a function's truth table, this can be easily reflected by a change in a bit vector storing the function's outputs. Furthermore, the effect of each operation is predictable, and so rules reflecting the changes for each operation can be predetermined and stored as templates. A further advantage of working in the functional domain is in the fact that all of the above operations can be performed very quickly as bit operations such as AND, OR, XOR, and SHIFT. Unfortunately, we do lose out on some opportunity to optimize over groups of functions, since we cannot get information about other functions from an individual function's truth table.

Table 2: An example of three functions each having the same number of true minterms in their output vectors.

$x_3x_2x_1$	$f_1(X)$	$x_3x_2x_1$	$f_2(X)$	$x_3x_2x_1$	$f_3(X)$
000	0	000	0	000	1
001	0	001	1	001	0
010	0	010	0	010	0
011	0	011	1	011	0
100	1	100	0	100	1
101	1	101	1	101	1
110	1	110	0	110	1
111	1	111	1	111	0

4.1. Optimizations. One optimization that can be made is to prefilter the functions. It is possible to group the functions based on the number of true bits in the output vector of the function's truth table. For example, the functions shown in Table 2 all have exactly four 1's, or four true *minterms* in their output vectors. Because the number of true bits is never changed by any of operations 1, 2, or 4 (permutation, negation of an input, or replacement of an input with an exclusive-or operation), we know that functions with differing numbers of true minterms can never be in the same class. This prefilter process categorizes the functions according to the number of true minterms in the output vector, resulting in $2^n + 1$ categories, of which two are trivial cases ($f(X) = 0$ and $g(X) = 1$). The effect of the Type 4 invariance operation is to negate the output, and so a function with k true minterms will be transformed to a function with $2^n - k$ true minterms. It can be shown that these will both fall into the same category, and so the prefilter can further reduce the number of categories to 2^{n-1}. Details of this proof are given in [30].

4.2. Rules. Our process of determining the classes then requires three steps: generate rules, prefilter, and then apply the rules. The rules describe how the output bits from the current function are remapped to create a new function within the same spectral class. For instance, if we consider the first invariance operation, permutation, then it is possible to permute n input variables in $n!$ possible ways resulting in $n!$ rules, including the original function. Table 3 illustrates these rules.

The second invariance operation, negation of input variables, results in 2^n possible rules, as there are 2^n ways in which combinations of the n input variables can be negated.

The third invariance operation, negation of the output, is something of a special case, as no swapping of output

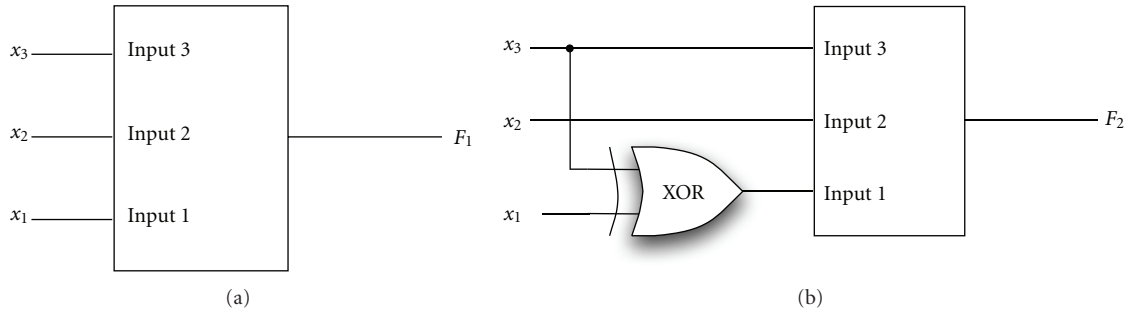

Figure 2: An example illustrating the Type 4 invariance operation, with (a) showing the original function, and (b) showing input 1 replaced with $x_3 \oplus x_1$.

Table 3: The six rules for the Type 1 operation, permutation, when applied to functions with $n = 3$ variables.

Rule	Permutation of Z							
0	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
1	z_0	z_2	z_1	z_3	z_4	z_6	z_5	z_7
2	z_0	z_1	z_4	z_5	z_2	z_3	z_6	z_7
3	z_0	z_2	z_4	z_6	z_1	z_3	z_5	z_7
4	z_0	z_4	z_1	z_5	z_2	z_6	z_3	z_7
5	z_0	z_4	z_2	z_6	z_1	z_5	z_3	z_7

Table 4: Type 4 input variable lookup table for $n = 3$. This table shows all possible exclusive-or combinations of the three input variables.

x_3	$x_3 \oplus x_2$	$x_3 \oplus x_1$	$x_3 \oplus x_2 \oplus x_1$
x_2	$x_2 \oplus x_1$	$x_2 \oplus x_3$	$x_2 \oplus x_1 \oplus x_3$
x_1	$x_1 \oplus x_2$	$x_1 \oplus x_3$	$x_1 \oplus x_2 \oplus x_3$

bits takes place. Thus there is no real rule generated for this operation.

The fourth invariance operation, replacement of a variable with an exclusive-or operation involving that variable, is a more difficult proposition. Figure 2 illustrates how an input whose original value was variable x_1 can be replaced with the exclusive-or operation $x_3 \oplus x_1$. Rules for this operation must list all possible legal replacements of the variables, keeping in mind that the replacement of a particular variable must incorporate the original variable. To generate a complete list, first all possible replacements are listed and placed in a lookup table such as is shown in Table 4. To generate a possible Type 4 operation rule, one must then choose one item from each row in Table 4, and this is repeated until all possible combinations have been realized. This will generate $(2^{n-1})^n$ combinations. However, the problem is that some of these combinations are invalid. For example let us take a function such as $f_1(X) = x_3 + x_2x_1$ and perform the following replacements:

$$\begin{aligned} &\text{replace } x_3 \text{ with } x_3 \oplus x_2 \oplus x_1, \\ &\text{replace } x_2 \text{ with } x_3 \oplus x_2 \oplus x_1, \\ &\text{replace } x_1 \text{ with } x_3 \oplus x_2 \oplus x_1. \end{aligned} \tag{4}$$

Table 5: The truth tables for $f_1(X) = x_3 + x_2x_1$ and for $f_2(X) = x_3 \oplus x_2 \oplus x_1$.

$x_3x_2x_1$	f_1	f_2
000	0	0
001	0	1
010	0	1
011	1	0
100	1	1
101	1	0
110	1	0
111	1	1

Then the resulting function becomes

$$\begin{aligned} f_2(X) &= [x_1 \oplus x_2 \oplus x_3] + [(x_1 \oplus x_2 \oplus x_3)(x_1 \oplus x_2 \oplus x_3)] \\ &= [x_1 \oplus x_2 \oplus x_3] + [x_1 \oplus x_2 \oplus x_3] \\ &= x_1 \oplus x_2 \oplus x_3. \end{aligned} \tag{5}$$

It is clear from examining the truth tables of these two functions as shown in Table 5 that they have different numbers of true minterms, and so cannot be in the same class.

To solve this problem, the replacements can be represented as a matrix in which each row represents a variable and each column indicates whether the variable is included in the replacement. Using this representation for functions $f_1(X)$ and $f_2(X)$ from the above example would result in matrices as shown in Figure 3. In this form, each combination must have true bits on the diagonal. To then separate the valid operations from the invalid combinations, a linear independence check is performed on the vectors of each input combination (the rows of the matrix for each combination of replacements). If a combination is found to be linearly independent, it is added to the list of valid Type 4 operations.

The final invariance operation is the replacement of the output $f(X)$ with $f(X) \oplus x_i$. This operation was not included in this work, so as to generate a list more easily comparable to work in [3].

$$\begin{array}{c} \\ x_3 \\ x_2 \\ x_1 \end{array}\begin{array}{c} \begin{array}{ccc} x_3 & x_2 & x_1 \end{array} \\ \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ x_3 \\ x_2 \\ x_1 \end{array}\begin{array}{c} \begin{array}{ccc} x_3 & x_2 & x_1 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \end{array}$$

(a) (b)

Figure 3: Matrices representing (a) the starting function $f_1(X) = x_3 + x_2x_1$ and (b) the function $f_2(X)$ resulting from replacing each variable with the network $x_3 \oplus x_2 \oplus x_1$.

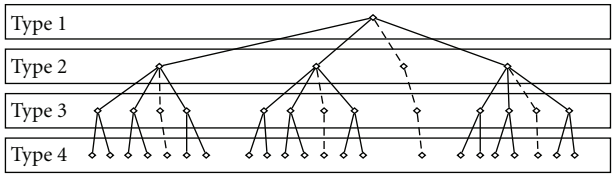

Figure 4: Tree illustrating the concept of combining all the transformation rules.

4.3. Applying the Rules. Simply applying each of the rules for each rule type to some chosen starting function will not achieve the desired result of producing all possible functions in a spectral class. To create all possible functions *all* of the rules must be considered in combination, and the eventual result must combine all rules in all possible combinations. Figure 4 shows a tree illustrating the concept of combining the transformation rules. Each result of the Type 1 rules must be operated on by each of the Type 2 rules. For each result of the Type 2 rules, the Type 3 operations must be applied. Finally, for each Type 3 result, each Type 4 operation must be applied. The leaf nodes of the tree represent all possible functions that can be realized from the original starting function. The first leaf node, when considering a postorder traversal, represents the starting function: in other words the original Boolean switching function, unmodified. The code for implementation of the prefilter, rule-generation, and application of the rules is available in [30].

5. Results & Analysis

5.1. A Correction to Previous Results. In [3] it was stated that there are 191 spectral classes to represent all functions for $n = 5$. Our research, however, indicates that several classes may inadvertently have been combined and that there are in fact 206 spectral classes needed to represent all 2^{2^5} functions. Table 6 shows the number of classes for $n = 1$ through 5 as determined by our approach. All previously published results agree with these values up to $n = 5$. As classifying all functions is a problem that grows not just exponentially but double-exponentially as the value of n increases, this problem must be optimized in order to compute a solution, and a computational solution is not easily checked. Careful analysis of the optimizations in this implementation indicates that 5.84×10^{11} transformations are required to classify all 2^{2^5} functions. Although this is still a lot of computation, it is significantly less than the 1.22×10^{19} transformations that must be performed when no optimization is added to the problem. It is thus necessary to present arguments demonstrating the accuracy of our results, since computational verification is nearly impossible.

Table 6: Table showing the number of spectral classes for $n = 1$ to 5.

n	no. classes our approach	no. classes from [3]	total no. functions
1	1	1	4
2	2	2	16
3	6	6	256
4	18	18	65536
5	206	191	4.29×10^9

Given this, we feel that the evidence of our correctness is substantial. When we examine previous computations of classes for $n = 3$ and $n = 4$, our results agree with previous work, providing one form of verification, at least for smaller values of n. Additionally, if we list the 191 classes found in previous work for $n = 5$, our research has resulted in the same 191 classes, although with an additional 15 new classes. These 15 new classes have a spectral signature matching classes contained within the other 191, so it is likely that in previous research the optimization techniques used in the implementation lead to the inadvertent combination of multiple classes. Since our optimizations rely strictly upon patterns that can be proven for a general n this is unlikely to be a factor in our results. In addition, a number of internal checks on our data and computations were performed, details of which are given in [30].

5.2. The Difficulties of $n \geq 5$. This is a difficult problem, as clearly solutions for small values of n such as 3, or 4 are relatively easy to determine. This is due to the nature of the numbers, as we must examine on the order of 2^{2^n} functions. For $n = 3$, we have $2^{2^3} = 256$, and for $n = 4$ we have $2^{2^4} = 65,536$. However, for $n = 5$ we have $2^{2^5} = 429,967,296$, and we have used the term double-exponential for this growth. Because of this growth, we suspect that as n increases there may be different "behaviours" of the coefficients. For instance, using the signature of the function was sufficient to classify functions for $n = 3$, but not for $n = 4$, so it is likely that optimizations of this nature used in computing the $n = 4$ classes may not have been suitable for the $n = 5$ classes. There are a few factors that may indicate why this might be. It seems as though the coefficients may "behave" differently for $n > 4$. As the value of n increases, there are more ways to combine the truth values of each function. It is possible that not all combinations of the four operation types are needed to compute all of the classes for smaller values of n. Indeed, it was previously thought that the signature of a function's coefficients was unique to a class, but as computation for higher values of n became feasible this was shown to be untrue. It is also possible that the spectral classes for $n < 5$ are in fact special cases and that it is not possible to observe any patterns without knowing the classes

Table 7: The numbers of rules for varying values of n.

	n	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
Type 1	$n!$	1	2	6	24	120	720
Type 2	2^n	2	4	8	16	32	64
Type 4	n/a	1	3	34	1688	370,752	347,638,784

for $n > 5$. Once spectral classes for $n > 5$ are calculated, it may appear that there are different patterns for even and odd values of n. Unfortunately, until the spectral class structure for higher values of n are computed, the answers to these questions will remain unknown.

5.3. Analysis of the Number of Rules. It is possible to calculate the number of rules required to transform a starting function into all other possible functions within that class. The number of rules for $n \leq 6$ is given in Table 7.

For Type 1 transformations, $n!$ rules are needed to produce all possible permutation of the input variables while Type 2 transformations require 2^n rules. Type 3 transformations do not have rules, as defined in Section 4.2 but rather the outputs of the function are negated. Therefore, for all values of n, the number of transformations for Type 3 is simply the constant 2. Type 4 transformations are somewhat more complex as there is no known general case to calculate the required number of rules, and a generalized equation for the number of rules for Type 4 is currently unknown and as such, an area of future research. In general, only an upper bound of $(2^{n-1})^n$ can be specified, which includes invalid transformations, thus we know only that the removal of invalid transformation will reduce the total possible number of rules. In Table 7 we show this as the equation n/a where a is yet to be determined.

Although the number of rules generated by this implementation for rules of Types 1 and 2 are confirmed, Type 4 cannot be compared if no general case can be provided. To confirm the number of Type 4 rules, an alternate method of generating the rules was created using Maple [31]. In this method all possible combinations of input variables for each value of n were checked for linear independence. Each set of input variables was checked using the determinant function that is built into Maple. If the result of the determinant modulo 2 does not equal 0 then it is considered to be linearly independent, and therefore valid combination of input variables. For $n \leq 5$ the number of linearly independent sets returned from Maple equaled the number of Type 4 rules generated by our implementation.

5.4. General Complexity. Spectral classification of Boolean functions is a very large problem and the number of transformations that must take place can be described as $A \cdot B \cdot C \cdot D \cdot E$ where A represents the total number of functions to be considered, B represents the number of Type 1 transformations, C represents the number of Type 2 transformations, D represents the number of Type 3 transformations, and E represents the upper bound for Type 4 transformations. Because E includes possibly invalid transformations, this expression represents an upper bound. The implementation of this approach is highly dependent on the order in which A through E are combined. In this implementation, for each item saved in A, the work associated with parts B, C, D, and E can be completely avoided. This is true for every term going from left to right in the expression. In other words, for each item saved in B, work in C, D, and E is avoided and so on.

Alternatively, we can consider this expression as a tree where Figure 4 represents the terms B, C, D, and E. In this tree, Figure 4 is the child node for each item in A. If there are 2^{2^n} starting functions with A, it means that Figure 4 must be traversed 2^{2^n} times; once for each item of A. The optimization approaches in this section are simply methods to prune this tree. The further up the tree these optimizations occur, the larger the overall benefit to the problem.

The analysis of the problem is first considered in its worst case. In the worst case, the total number of function transformations that must be performed is as follows:

$$\begin{aligned} A &: \quad 2^{2^n} \\ B &: \quad n! \\ C &: \quad 2^n \\ D &: \quad 2 \\ E &: \quad (2^{n-1})^n. \end{aligned} \tag{6}$$

There are thus $2^{2^n} \cdot n! \cdot 2^n \cdot 2 \cdot (2^{n-1})^n$ transformations that must be performed to spectrally classify all functions of n input variables.

The first reduction of this is fairly straightfoward; simply, the number of starting functions can be reduced by half by observing that the second half of all 2^{2^n} functions can be achieved by a Type 3 operation applied to the first 2^{2^n-1} functions. There are then $2^{2^n-1} \cdot n! \cdot 2^n \cdot 2 \cdot (2^{n-1})^n$ transformations in the optimized general case for functions of n input variables.

Since all possible combinations of all four spectral transformations are applied to a starting function, all functions that exist within the same class as the starting function are also discovered. This observation further reduces the number of starting functions from 2^{2^n-1} to the number of spectral classes, S_n. As a result of the second optimization to A, the number of transformations in the optimized general case is now $S_n \cdot n! \cdot 2^n \cdot 2 \cdot (2^{n-1})^n$. For $n = 5$, the total number of transformations is as follows:

$$\begin{aligned} A &: \quad 206 \\ B &: \quad 5! \\ C &: \quad 2^5 \\ D &: \quad 2 \\ E &: \quad 1,048,576. \end{aligned} \tag{7}$$

We can see that the optimizations and pruning as described above and in Section 4.1 have reduced the number of transformations considerably, and the final number of transformations required to classify all functions for $n = 5$ is 1.66×10^{12}. This is significantly smaller than the brute force case that would require 1.22×10^{19} transformations.

5.5. Future Considerations. It has already been shown that classification of all 2^{2^n} functions is an incredibly difficult problem, especially as n increases to values above 4. For $n > 5$, it is impractical to use current methods of classification; therefore some other method is needed to calculate these classes. One approach is to use existing data from smaller values of n and extrapolate to higher values. Using prediction of this nature, it may be possible to derive all, or a large portion, of the classes for $n + 1$ variables. This could greatly decrease the amount of processing needed and could make classification for $n > 5$ feasible.

When considering the first 128 functions for $n = 3, 4$, and 5 many of the classes remain the same as the value of n increases. This is an interesting observation that deserves further study.

We note that the program for this work was initially written in Java, and then ported to C++ when we ran into speed and performance issues. The computation was carried out on a Macintosh G5 with 2.5 GHz dual processors and 6 GB of RAM. On this platform, processing of the $n = 5$ classes required between one and two days (24 to 48 hours) of processing time to complete. As platforms and language support improve, it may be possible to continue improving this; unfortunately as the numbers increase for $n = 6$ even significant improvements will not provide enough speed-up to complete in a reasonable amount of time, thus some type of derivation or prediction process is likely to be required.

6. Conclusion

This work was begun with the goal of continuing work begun in [3] and by other researchers, and extending the computational results to higher values of n. Instead, what we found was that even with faster technology and larger amounts of memory the problem of determining the spectral classes for Boolean functions grows too quickly. Moreover, we determined that there is significant evidence that previous implementations for determining the classes for $n = 5$ were, in fact, faulty, and resulted in incorrect results.

This research focused on an entirely new approach to the computation of the spectral coefficients, and found that an additional 15 classes should be added to the previously computed lists. In addition, our optimization techniques have been well documented and closely checked, leading us to have a great deal of confidence in them. This will, we hope, provide a basis for future researchers to build upon without having to rebuild our work. These optimizations include the design of a prefilter to reduce the numbers of functions on which our technique must operate. We also have taken the step of mathematically proving for a general n any optimations that were utilized.

It is our goal to eventually determine a prediction method for higher values of n, so that rather than computing all spectral classes we could instead predict, based on existing data, which class a function might fall in to. In addition we conjecture that, once enough data is available, we may be able to extrapolate the structure and composition of classes of functions for higher values of n based on data for lower values.

Acknowledgment

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for their support of this work.

References

[1] C. R. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis," *IEEE Transactions on Computers*, vol. 24, no. 1, pp. 48–71, 1975.

[2] M. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley & Sons, New York, NY, USA, 1976.

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

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

[5] J. E. Rice, *Autocorrelation coefficients in the representation and classification of switching functions*, Ph.D. thesis, University of Victoria, 2003.

[6] M. A. Thornton and R. Drechsler, "Computation of spectral information from logic netlists," in *Proceedings of the 30th IEEE International Symposium on Multiple-Valued Logic (ISMVL '00)*, pp. 53–58, May 2000.

[7] M. A. Thornton and V. S. S. Nair, "Efficient spectral coefficient calculation using circuit output probabilities," *Digital Signal Processing*, vol. 4, no. 4, pp. 245–254, 1994.

[8] M. A. Thornton and V. S. S. Nair, "Efficient calculation of spectral coefficients and their applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1328–1341, 1995.

[9] A. Žužek, R. Drechsler, and M. A. Thornton, "Boolean function representation and spectral characterization using AND/OR graphs," *Integration, the VLSI Journal*, vol. 29, no. 2, pp. 101–116, 2000.

[10] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 54–60, June 1993.

[11] D. Jankovic, R. S. Stankovic, and R. Drechsler, "Decision diagram method for calculation of pruned Walsh transform," *IEEE Transactions on Computers*, vol. 50, no. 2, pp. 147–157, 2001.

[12] R. S. Stanković, M. Stanković, C. Moraga, and T. Sasao, "Calculation of Reed-Muller-Fourier coefficients of multiple-valued functions through multiple-place decision diagrams," in *Proceedings of the International Symposium on Multiple-Valued Logic (ISMVL '94)*, pp. 82–88, 1994.

[13] S. Purwar, "An efficient method of computing generalized Reed-Muller expansions from binary decision diagram," *IEEE Transactions on Computers*, vol. 40, no. 11, pp. 1298–1301, 1991.

[14] M. A. Thornton and R. Drechsler, "Spectral decision diagrams using graph transformations," in *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE '01)*, pp. 713–719, 2001.

[15] M. A. Thornton, "Mixed-radix MVL function spectral and decision diagram representation," *Automation and Remote Control*, vol. 65, no. 6, pp. 1007–1017, 2004.

[16] B. J. Falkowski and T. Sasao, "Unified algorithm to generate Walsh functions in four different orderings and its

programmable hardware implementations," *IEE Proceedings: Vision, Image and Signal Processing*, vol. 152, no. 6, pp. 819–826, 2005.

[17] B. J. Falkowski, "Parallelization of methods to calculate Walsh spectra for logic functions," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 10, no. 2, pp. 91–127, 2004.

[18] J. P. Hansen and M. Sekine, "Synthesis by spectral translation using Boolean decision diagrams," in *Proceedings of the 33rd Annual Design Automation Conference*, pp. 248–253, June 1996.

[19] J. Moore, K. Fazel, M. A. Thornton, and D. M. Miller, "Boolean function matching using Walsh spectral decision diagrams," in *IEEE Dallas ICAS Workshop on Design, Applications, Integration and Software (DCAS '06)*, pp. 127–130, Richardson, TX, USA, October 2006.

[20] D. M. Miller, "Spectral and two-place decomposition techniques in reversible logic," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems (MWSCAS '02)*, vol. 2, pp. 493–496, August 2002.

[21] 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.

[22] M. G. Karpovsky, R. S. Stanković, and C. Moraga, "Spectral techniques in binary and multiple-valued switching theory: a review of results in the decade 1991–2000," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 10, no. 3, pp. 261–286, 2004.

[23] M. A. Thornton, R. Drechsler, and D. M. Miller, *Spectral Techniques in VLSI CAD*, Kluwer Academic Publishers, 2001.

[24] B. J. Falkowski and S. Yan, "Properties of logic functions in spectral domain of sign Hadamard-Haar transform," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 11, no. 1-2, pp. 185–211, 2005.

[25] B. J. Falkowski and S. Yan, "Ternary Walsh transform and its operations for completely and incompletely specified Boolean functions," *IEEE Transactions on Circuits and Systems I*, vol. 54, no. 8, pp. 1750–1764, 2007.

[26] C. C. Tsai and M. Marek-Sadowska, "Boolean functions classification via fixed polarity Reed-Muller forms," *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 173–186, 1997.

[27] A. B. Lapshin, "Classification of Boolean functions by the invariants of their matrix representation," *Automation and Remote Control*, vol. 67, no. 7, pp. 1100–1107, 2006.

[28] J. E. Rice and J. C. Muzio, "Use of the autocorrelation function in the classification of switching functions," in *Proceedings of the Euromicro Symposium on Digital System Design: Architectures, Methods and Tools (DSD '02)*, pp. 244–251, 2002.

[29] I. Strazdins, "Universal affine classification of Boolean functions," *Acta Applicandae Mathematicae*, vol. 46, no. 2, pp. 147–167, 1997.

[30] N. Anderson, *The classification of Boolean functions using the Rademacher-Walsh transform*, M.S. thesis, University of Victoria, 2007.

[31] Maplesoft, "Maple 10," 2007, http://www.maplesoft.com/products/maple/history/documentation.aspx.