A Spectral Algorithm for Ternary Function Classification

D. Michael Miller
Department of Computer Science
University of Victoria
Canada
Email: mmiller@uvic.ca

Mathias Soeken
École Polytechnique Fédérale de Lausanne
Switzerland
Email: mathias.soeken@epfl.ch

***Abstract*—The spectral representation and classification of 2-valued and multiple-valued functions has been previously studied and found to be useful in logic design and testing for conventional circuits. Spectral techniques also have potential application for reversible and quantum circuits. This paper addresses the classification of ternary functions into spectral translation equivalence classes. An efficient algorithm is presented that determines the spectral translations to map a given function to the representative function for the equivalence class containing the given function. Using this algorithm we show that the 2-variable ternary functions partition into 11 equivalence classes. While the number of spectral equivalence classes for ternary functions with 3 or more variables is very large, prohibiting full enumeration, we determine a lower bound of 167,275 classes for 3 variables. The algorithm can be used for a significant number of variables to quickly determine if two functions fall within the same equivalence class and, if they do, to find a sequence of spectral translations to map one to the other. Generalization of the approach to higher radix functions is briefly discussed.**

I. Introduction

The classification of 2-valued functions has been well studied beginning with NPN classification [1] which partitions functions into equivalence classes based on negation of variables, permutation of variables, and negation of the function. Spectral techniques have also been used for 2-valued function classification [2]–[5] where two spectral translations augment the NPN operations. Using the spectral approach the set of all n-variable 2-valued functions is partitioned into significantly fewer equivalence classes. These classification schemes have been employed in logic design, technology mapping, testing and other applications.

Karpovsky [6] provided a comprehensive development of the extension of spectral techniques to multiple-valued functions including spectral translations (see also [7]). Hurst [2] and Moraga [8] considered the application of spectral techniques to the classification of multiple-valued functions with particular emphasis on ternary functions.

In this paper, we consider the spectral classification of ternary functions. The principal contributions are the introduction of an ordering of ternary functions that allows us to define a unique representative function for a spectral equivalence class, and based on that concept, a novel algorithm to map a function to the representative function for the equivalence class containing the given function. Using this algorithm we identify 11 spectral equivalence classes for 2-variable ternary functions and estimate the number of spectral equivalence classes for 3-variable ternary functions to be at least 167,275.

Our approach has two important aspects. First, even though the number of equivalence classes grows very large for ternary functions and tabulating them is impractical except for $n < 3$, the presented algorithm can be used to identify if two functions are in the same class, *i.e.*, they can be translated to the same representative function. Second, the algorithm identifies the sequence of translations to map a function to the representative function and since the translations are all invertible, our algorithm can identify a single sequence of translations to map one function to another in the same equivalence class, even for larger n.

Function classification is an interesting problem in its own right because it increases the understanding of functions and their interrelations. In addition because spectral translations and spectral equivalence is $\bmod r$ sum based (XOR in the 2-valued case), the presented algorithm has potential applications in cryptography [9]–[11], reversible circuits [12], quantum computation [13] and arithmetic verification [14].

The rest of the paper is organized as follows. Section II provides the necessary background on ternary functions and their spectra. Spectral translations are defined and the idea of function classification is discussed in Section III including the concepts of function ordering and representative functions for spectral equivalence classes. Our new algorithm is presented in Section IV and experimental results are given in Section V. Section VI concludes the paper with observations and discussion of future work including a brief description of the challenge of extending the approach to higher radix functions.

II. Ternary Functions and their Spectra

An n-input ternary function $f(x_1, x_2, \ldots, x_n)$ is a mapping $f : \{0,1,2\}^n \rightarrow \{0,1,2\}$. Such an f can be represented by a value (column) vector, denoted F, with 3^n entries. In this work, we use a function coding where 0, 1 and 2 are represented by $a^0 = 1$, a^1 and a^2, respectively, where $a = \frac{1}{2}(-1 + \sqrt{3}\mathrm{i})$, $\mathrm{i} = \sqrt{-1}$. Note that $a^2 = \frac{1}{2}(-1 - \sqrt{3}\mathrm{i})$, the complex conjugate of a.

In the developments below, it will be seen that all values encountered take the form $x_0 + x_1 a + x_2 a^2$ where the x_i are nonnegative integers. For convenience, we shall denote

$x_0 + x_1 a + x_2 a^2$ by the ordered triple $[x_0, x_1, x_2]$. Since $a + a^2 = -1$, it follows that $1 + a + a^2 = 0$ so we have the notion that any value $x_0 + x_1 a + x_2 a^2$ can be *reduced* so that at least one $x_i = 0$. Reduction involves finding the minimum of the x_i and subtracting that value from each of the x_i.

It follows directly from the definition of a, that $a^p = a^q$, if $q = p \bmod 3$. Since $a^0 = 1$ we need therefore only consider multiplication by a and by a^2.

The following operations are used below:

- addition: $[x_0, x_1, x_2] + [y_0, y_1, y_2] = [x_0 + y_0, x_1 + y_1, x_2 + y_2]$ which must be reduced as described above
- multiplication by a: $a \times [x_0, x_1, x_2] = [x_2, x_0, x_1]$
- multiplication by a^2: $a^2 \times [x_0, x_1, x_2] = [x_1, x_2, x_0]$
- complex conjugate: $\overline{[x_0, x_1, x_2]} = [x_0, x_2, x_1]$
- magnitude: $\|[x_0, x_1, x_2]\| = [x_0, x_1, x_2] \times \overline{[x_0, x_1, x_2]} = x_0^2 + x_1^2 + x_2^2 - x_0 x_1 - x_0 x_2 - x_1 x_2$
- greater than: $[x_0, x_1, x_2] > [y_0, y_1, y_2]$ if for the lowest i such that $x_i \neq y_i$, $x_i > y_i$.

The approach just described has significant computational advantage since, as noted above, complex numbers are represented by a triple of nonnegative integers. The rounding errors associated with other approaches, *e.g.*, floating-point, are thereby avoided.

Definition 1: The Chrestenson spectrum, also known as the Vilenkin-Chrestenson spectrum, [6], [15], [16] of a ternary function is given by

$$S = C^n F, \tag{1}$$

where the transform matrix C^n is defined by

$$C^n = \begin{bmatrix} C^{n-1} & C^{n-1} & C^{n-1} \\ C^{n-1} & a^2 C^{n-1} & a C^{n-1} \\ C^{n-1} & a C^{n-1} & a^2 C^{n-1} \end{bmatrix}, C^1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & a^2 & a \\ 1 & a & a^2 \end{bmatrix}.$$

For example for $n = 2$,

$$C^2 = \left[\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & a^2 & a & 1 & a^2 & a & 1 & a^2 & a \\ 1 & a & a^2 & 1 & a & a^2 & 1 & a & a^2 \\ \hline 1 & 1 & 1 & a^2 & a^2 & a^2 & a & a & a \\ 1 & a^2 & a & a^2 & a & 1 & a & 1 & a^2 \\ 1 & a & a^2 & a^2 & 1 & a & a & a^2 & 1 \\ \hline 1 & 1 & 1 & a & a & a & a^2 & a^2 & a^2 \\ 1 & a^2 & a & a & 1 & a^2 & a^2 & a & 1 \\ 1 & a & a^2 & a & a^2 & 1 & a^2 & 1 & a \end{array}\right].$$

It is readily verified that $(C^n)^{-1} = \frac{1}{3^n}(C^n)^*$ where * denotes conjugate transpose (note that C^n is symmetric so * in this case is simply the conjugate). As a consequence we observe that the Chrestenson spectrum of a ternary function is unique.

As an example of computing a spectrum, the function $f(x_1, x_2)$ shown in Table I is represented by the vector F of complex numbers. The spectrum computed using (1) is given by S in Table I. Note that due to the recursive structure of C^n, the matrix multiplication can be implemented as a fast transform [17]. The matrix multiplication has time complexity $O(3^{2n})$ whereas for the fast transform it is $O(n3^n)$.

x_2	x_1	f	F	p	S
0	0	0	[1,0,0]	0	[0,4,2]
0	1	1	[0,1,0]	1	[5,0,1]
0	2	2	[0,0,1]	2	[0,1,2]
1	0	1	[0,1,0]	3	[2,0,1]
1	1	1	[0,1,0]	4	[5,0,1]
1	2	2	[0,0,1]	5	[4,2,0]
2	0	2	[0,0,1]	6	[2,0,1]
2	1	1	[0,1,0]	7	[1,2,0]
2	2	1	[0,1,0]	8	[0,1,2]

TABLE I
EXAMPLE FUNCTION f, CODING F AND SPECTRUM S

The coefficients of the spectrum S are denoted s_p, $0 \leq p < 3^n$. The n-digit ternary expansion of p gives the value assignment to the variables associated with the coefficient. If the coefficients are arranged by increasing subscript value, as in Table I, we say the coefficients are in *natural* order. In this paper, we also employ an alternate ordering of the spectral coefficients which we term *weighted* order where the coefficients are arranged into groups by the number (ascending) of nonzero digits in the ternary expansions of the coefficient indices. Each group is itself arranged into subgroups by the number (ascending) of digits equal to 2. Within a subgroup the coefficients are by ascending order of the index values.

For $n = 2$, the weighted order is as follows where the indices are shown as 2-digit ternary numbers:

$$s_{00} || s_{01} s_{10} | s_{02} s_{20} || s_{11} | s_{12} s_{21} | s_{22}$$

Symbols || separate groups and | separate subgroups. For $n = 3$, the weighted order is:

$$s_{000} || s_{001} s_{010} s_{100} | s_{002} s_{020} s_{200} || s_{011} s_{101} s_{110} |$$
$$s_{012} s_{021} s_{102} s_{120} s_{201} s_{210} | s_{022} s_{202} s_{220} ||$$
$$s_{111} | s_{112} s_{121} s_{211} | s_{122} s_{212} s_{221} | s_{222}$$

III. SPECTRAL TRANSLATIONS AND FUNCTION CLASSIFICATION

Five spectral translations [2], [3] are used in the spectral classification of 2-valued functions. We here present extensions of those translations to the ternary case. For each we give the functional translation and the corresponding operation on the spectrum. Note that the extensions presented here differ from those given in [8] which assumed arbitrary permutations and linear translations to be available. Here we use sum $\bmod 3$ and value interchange as the basic operations. We also use variable interchange as used in [8].

In the following definition of the spectral translations, let $p = (p_1 p_2 \dots p_n)_3$ and $q = (q_1 q_2 \dots q_n)_3$ be two ternary numbers as found in the indexes of spectrum coefficients.

Translation 1 Interchange of input variables x_i and x_j. This corresponds to the interchange of the spectral coefficient pairs given by,

$$s_p \leftrightarrow s_q$$

where $q = (p_1 \dots p_{i-1} p_j p_{i+1} \dots p_{j-1} p_i p_{j+1} \dots p_n)$, *i.e.*, q is obtained from p by interchanging digits i and j.

In the 2-valued case, translation 2 involves the negation of an input variable. We expand this to two translations: (2a) cycle and (2b) value exchange.

Translation 2a Cycle of the input variable x_i by $v \in \{0,1,2\}$, *i.e.*, replace $x_i \oplus v$ for x_i. For each s_p, $p \neq 0$, multiply s_p by $a^{(vw) \bmod 3}$, where $w = p_i$.

Translation 2b Exchange the values v_a and v_b, $v_a, v_b \in \{0,1,2\}$ for input variables x_i. Let $m_1 = (v_a + v_b) \bmod 3$ and $m_2 = (3 - m_1) \bmod 3$. Construct a new spectrum S^* where $\forall p$

- if $p_i = 0$, $s^*_p = s_p$
- if $p_i = 1$, $s^*_q = s_p$ multiplied by a^{m_2} where $q = p$ except that $q_i = 2$
- if $p_i = 2$, $s^*_q = s_p$ multiplied by a^{m_1} where $q = p$ except that $q_i = 1$

In the 2-valued case, translation 3 involves the negation of the function. Again we expand this into two translations.

Translation 3a Cycle of the function by $v \in \{0,1,2\}$. Multiply every spectral coefficient by a^v.

Translation 3b Exchange of the values v_a and v_b, $v_a, v_b \in \{0,1,2\}$ for the function. Construct a new spectrum S^* where $s^*_q = s_p \forall p$ where q is p with all 1's changed to 2's and all 2's changed to 1's, and for each coefficient of S^* the v_a and v_b positions in the triple defining the coefficient are interchanged.

Translation 4 Replacement of input variable x_i by $x_i \oplus x_j$. Construct a new spectrum S^* with $s_q = s_p$ where $q = p$ except that $q_j = (p_i + p_j) \bmod 3$.

Translation 5 Replacement of the function f by $f \oplus x_i$. Construct a new spectrum S^* with $s_q = s_p$ where $q = p$ except that $q_i = (p_i + 1) \bmod 3$.

Translations 1, 2b and 3b are each clearly self-inverse. Translations 2a, 3a, 4 and 5 each involve the $\bmod 3$ sum and the inverse operation for each is to apply the same translation two more times.

Application of these translations leads to the following key concept:

Definition 2: Two functions $f(x_1, x_2, \ldots, x_n)$ and $g(x_1, x_2, \ldots, x_n)$ are termed *spectral translation equivalent* if f can be transformed into g by the application of some sequence of the above translations. Since the translations all have inverses, it is straightforward to identify a sequence of translations to transform g to f. Also as shown in the definitions of the translations they can be directly carried out in the spectral domain, *i.e.*, it is straightforward to transform between the spectra of f and g.

Spectral translation equivalence divides ternary functions into spectral equivalence classes leading to the idea of using spectral translation for ternary function classification. Such classification was introduced in [8].

We define an ordering of the spectra of n-variable ternary functions and relate that to spectral classification.

Definition 3: Given two n-variable ternary functions $f(x_1, x_2, \ldots, x_n)$ and $g(x_1, x_2, \ldots, x_n)$ with spectra S_f and S_g respectively, we say f *precedes* g, denoted $f \prec g$ if for the first coefficient position (in weighted order) for which the coefficients from S_f and S_g differ, the coefficient from S_f has larger magnitude, or if the two coefficients have the same magnitude, the coefficient from S_f is greater than the coefficient from S_g. Note that for convenience we will also write $S_f \prec S_g$.

Definition 4: Clearly, a spectral equivalence class must contain a function f^R that precedes every other function in the class. We term f^R the *representative function* for the class.

The problem addressed in this paper is:

Problem Statement: *Given a ternary function f (spectrum S) find a low cost sequence of spectral translations that transforms f to the representative function f^R (spectrum S^R) of the spectral equivalence class that contains f.*

The cost of a sequence of translations depends on the cost model used. If all translations are assumed to have unit cost, the cost is simply the number of translations. Assigning 0 cost to each translation means that any sequence leading to f^R is equally acceptable. In this paper we use the following:

Cost Model:

- Translation 1 interchanges 2 variables and requires a swap gate which can be implemented using three $\bmod 3$ sum gates. We thus assume a cost of 3.
- Translations 2a, 2b, 3a and 3b each require a single gate, cycle or value exchange to implement an inversion and we thus use a cost of 1.
- Translations 4 and 5 each require a single $\bmod 3$ sum gate and again we use a cost of 1.

IV. Transformation Algorithm

The transformation algorithm presented in this section is based on a 2-valued version developed by the authors and presented in [18]. The ternary version presented here is not a simple extension. Complications arise due to the complexity of extending 2-valued NOT to ternary unary operations as noted above in the extension of the spectral translations. In addition, procedure ADJUST has no counterpart in the 2-valued case.

Procedure TRANSFORM, introduced in this paper, maps a function f, with spectrum S, to the representative function f^R, with spectrum S^R, for the equivalence class that contains f. In doing so it finds a low cost sequence of translations but does not necessarily find the minimal cost sequence as the algorithm does not search all possible translation sequences. The parameters to the procedure are the spectrum S, the number of function variables n, and a third parameter v explained below. TRANSFORM is recursive and performs a non-exhaustive search of translation possibilities.

When the procedure completes the result is the spectrum S^R and the sequence of translations is in T^R. T is used to record a sequence of translations as it is built. S^R, T^R and T are for simplicity and efficiency global to the procedures.

TRANSFORM uses a procedure ADJUST that performs the final adjustments to a spectrum. ADJUST is recursive because of the myriad of choices of translations to finalize a spectrum. ADJUST in turn uses a procedure called PRECEDES that accepts a spectrum, parameter S, and compares it to the spectrum S^R which as noted above is a global. S^R is replaced by S and the sequence of translations T replaces T^R if $S \prec S^R$, see Definition 3.

TRANSFORM and ADJUST employ seven procedures, TRANS1, TRANS2A, ..., TRANS5, one for each of the spectral translations described in Section III. Each applies a translation to the parameter spectrum S. The translation is also appended

to the end of the global T. Note that it is important that T is maintained in the order the translations are to be applied.

We use the notation $j[p]$ to mean the p^{th} digit (0 is the least significant position) in the ternary expansion of j.

```
1: procedure TRANSFORM(S, n, v)
2:     if v = 0 then
3:         S^R ← S
4:         T^R ← φ
5:     end if
6:     if v ≤ n then
7:         find min and max magnitude coefficient values
8:             consider all coefficients if v = 0
9:             otherwise consider all s_p beginning at p = 3^(v−1)
10:    end if
11:    if v > n or max = 0 then
12:        ADJUST(S, n, 0)
13:        return
14:    end if
15:    for each s_j (in weighted order) starting at
16:              j = 0 if v = 0
17:              or j = 1 otherwise do
18:        if ||s_j|| = max then
19:            S^1 ← S
20:            if v = 0 then
21:                T ← φ
22:            end if
23:            save ← len(T)
24:            if j ≠ 0 then
25:                k ← (lowest p ≥ v such that
26:                        j[p − 1] > 0
27:                for each 1 ≤ p ≤ n, p ≠ k do
28:                    if j[p] > 0 then
29:                        TRANS4(S^1, n, k, p)
30:                        if j[p] = j[k] then
31:                            TRANS4(S^1, n, k, p)
32:                        end if
33:                    end if
34:                end for
35:                if v = 0 then
36:                    TRANS5(S^1, n, k)
37:                    if j[k] = 1 then
38:                        TRANS5(S^1, n, k)
39:                    end if
40:                else
41:                    if k ≠ v then
42:                        TRANS1(S^1, n, k, v)
43:                    end if
44:                end if
45:            end if
46:            TRANSFORM(S^1, n, v + 1)
47:            if v = 0 and min = max then
48:                return
49:            end if
50:            len(T) ← save
51:        end if
52:    end for
53:    return
54: end procedure
```

TRANSFORM implements a recursive search of depth $n+1$ that chooses appropriate translations to assign in order values to S_0^R followed by the first order coefficients in order from variable x_1 to x_n. Parameter v identifies which variable, and therefore which coefficients of S^R, are under consideration so the initial call should be TRANSFORM$(S, n, 0)$. The operation of TRANSFORM is as follows:

- lines 2–5: $v = 0$ is the top of the search so S^R is initialized to S, the spectrum of interest, and T^R, the sequence of translations to map S to S^R, is set to empty.
- lines 6–10: The min and max magnitude coefficient values are found by a simple linear search. All coefficients are considered if $v = 0$, otherwise the coefficients beginning with the first coefficient with a nonzero digit in position $v-1$ of the index are considered. This operation is skipped if $v > n$ which is the terminal case of the recursion.
- lines 11–14: This is the terminal case for the TRANSFORM recursion which is when $v > n$ or when $max = 0$. ADJUST is called to apply appropriate type 2 and 3 translations to the spectrum, see details below.
- lines 15–52: The coefficients are considered in weighted order. S_0 is only considered if $v = 0$ which is the case for choosing the appropriate value for S_0^R.
  - 18: Consider the coefficients where $||s_j|| = max$.
  - 19–22: Make S^1 a copy of S and if $v = 0$ (the top of the recursion) set the sequence of translations T to empty. S^1 is needed so that upon return from a recursion S is unchanged.
  - 23: Save the current length of T. This is necessary to restore the sequence of translations upon return from a recursion.
  - 24–45: If $j \neq 0$ translations may be required to move S_α^1 to S_v^1.
    - 25–34: Set k to be the lowest $p \geq v$ such that the p^{th} digit in the ternary expansion of j is nonzero and then apply type 4 translations to move S_j^1 to S_k^1.
    - 35–39: If $v = 0$, a type 5 translations are applied to move S_k^1 to S_0^1.
    - 41–43: Else if $k \neq v$, apply a type 1 translation to move S_k^1 to S_v^1
  - 46: This is the recursive call to transform S^1 for $v+1$, the next level of recursion.
  - 44–46: At the top of the recursion ($v = 0$) this avoids excessive searching of a 'flat' spectrum ($min = max$) which is when all coefficients have the same magnitude .
  - 50: Restore the length of T to its value before the last coefficient considered in preparation for the next S_α to be considered.
- line 53: The recursion options are exhausted so return.

The following procedure applies Definition 3.

```
1: procedure PRECEDES(S, n)
2:     find the first j (in weighted order) such that s_j ≠ s_j^R
3:     if there is no such j and cost(T) < cost(T^R) or
4:             ||s_j|| > ||s_j^R|| or
5:             ||s_j|| = ||s_p^R|| and s_j > s_j^R then
6:         S^R ← S
7:         T^R ← T
8:     end if
9:     return
10: end procedure
```

Procedure ADJUST refers to normalizing a spectral coefficient where a coefficient represented by $[x_0, x_1, x_2]$ is normalized if $x_0 \geq x_1 \geq x_2$ recalling that all three values cannot be equal.

```
1: procedure ADJUST(S, n, v)
2:     if v > n then
3:         PRECEDE(S, n)
4:         return
5:     end if
6:     S^1 ← S
7:     if v = 0 then
8:         TRANS3A(S^1, n, q) where q is the
9:                 rotation factor to normalize s_0
10:        if s^1_01 < s^1_02 then
11:            TRANS3B(s^1, n, 1, 2)
12:        end if
13:        adjust(s^1, n, 1)
14:        if s^1_01 = s^1_01 then
15:            TRANS3B(s^1, n, 0, 1)
16:            ADJUST(s^1, n, 1)
17:        end if
18:        if s^1_01 = s^1_02 then
19:            TRANS3B(s^1, n, 1, 2)
20:            ADJUST(s^1, n, 1)
21:        end if
22:    else
23:        save ← len(T)
24:        TRANS2A(S^1, n, v, q) where q is the
25:                rotation factor to normalize s^1_{3^{v-1}}
26:        ADJUST(S^1, n, v + 1)
27:        if ||S^1_{2×3^{v-1}}|| ≥ ||S^1_{3^{v-1}}|| then
28:            TRANS2B(S^1, n, v, 1, 2)
29:            TRANS2A(S^1, n, v, q) where q is the
30:                rotation factor to normalize s^1_{3^{v-1}}
31:            ADJUST(S^1, n, v + 1)
32:        end if
33:        len(T) ← save
34:    end if
35: end procedure
```

ADJUST is recursive and operates as follows:

- lines 2–5: This is the terminal case for the ADJUST recursion. PRECEDES is called and if $S \prec S^R$ or $S = S^R$ and $cost(S) < cost(S^R)$, S will be copied into S^R and T will be copied into T^R. Return because this is the terminal case in the recursion.
- line 6: Make S^1 a copy of S.
- lines 7–21: For $v = 0$ type 3a and type 3b translations are applied to adjust s^1_0. This consists of normalization (lines 8–9) and selected value interchanges (lines 10–21). Note the multiple recursive calls to ADJUST to consider the full possibilities of transformation affecting s^1_0.
- lines 22-34: For $0 < v \leq n$, this section applies appropriate type 2a and 2b translations to adjust $s^1_{3^{v-1}}$. As above, multiple recursive calls to ADJUST are required to explore all the options. Also note that the length of the translations sequence is saved and restored as was required in TRANSFORM.

V. Experimental Results

Our first experiment was to apply the transformation algorithm to the $3^{3^2} = 19,683$ two variable functions and to determine how many unique representative functions (equivalence classes) are found. Our implementation is written in C and applying it to all 19,683 functions took 0.78 CPU sec. on a laptop computer with a dual core 2.5 GHz Intel i5 processor.

The results are shown in Table II. Eleven classes were found. For each class, we show

- the count of the number of functions in the class,
- the representative function coded as a decimal number with f^R_0 being the least significant digit in the ternary expansion of that number, and
- the spectrum of the representative function.

It is interesting to note that the sizes (number of functions) of the classes are quite varied. We also observe that considering the 19,683 functions, the maximum transition cost for a single function was found to be 12, and the average over all functions was found to be 6.003.

Note that 12 classes were identified in [8] but by applying our method to example functions from [8] we have found that the classes identified as 8 and 10 in that work map to a single class which is class 7 in Table II. The transformations are shown in Table III.

For $n = 3$, there are $k = 3^{3^3} = 7,625,597,484,987$ ternary functions and an exhaustive search of all functions is intractable. In this case, we chose to sample random functions. In particular, the search scheme starts from function 0 and chooses each next function by adding a pseudo-random number to the current function number. The pseudo-random numbers are generated using the C library function rand(). For the system used, RAND_MAX = 32,767 so on average the jump from one function to the next is 16,383.5. A single application of this scheme will thus on average consider $k/16,383.5 = 465,443,738.2$ functions and for the computer used takes about 19.7 CPU hours.

We have applied the scheme five times using C library function srand() to seed the random number generator to produce a different set of random functions for each trial. Over the five trials, 167,275 unique representative functions were found although no single trial found them all. As seen above for the case of $n = 2$, the size of the classes is quite varied and some are relatively small making them less likely to be found by processing randomly selected functions. The closest in a single trial was 167,272. Also note that in each of the trials all the representative functions were found by approximately half way through the trial.

The 167,275 representative functions suggest the number of spectral equivalence classes for $n = 3$ but it can only be considered a lower bound as we only checked randomly selected functions. While this seems a very large number of classes we note that $167,275/3^{3^3} = 2.194 \times 10^{-8}$. For comparison it is known there are 48 spectral equivalence classes for the $2^{2^5} = 4,294,967,2964$ 2-valued functions giving a ratio of 1.118×10^{-8}.

VI. Conclusion and Future Work

This paper has presented a novel algorithm which maps a ternary function to the representative function for the spectral

TABLE II
SPECTRAL CLASSES FOR $n = 2$

class	count	f^R	00	01	10	02	20	11	12	21	22
1	27	0	[9,0,0]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]
2	486	81	[8,1,0]	[2,1,0]	[2,1,0]	[1,0,2]	[1,0,2]	[1,0,2]	[0,2,1]	[0,2,1]	[2,1,0]
3	1944	2268	[7,2,0]	[4,2,0]	[2,0,1]	[2,0,4]	[2,0,1]	[0,1,2]	[1,2,0]	[0,1,2]	[1,2,0]
5	216	2271	[6,3,0]	[6,3,0]	[0,0,0]	[3,0,6]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]
4	3888	111	[6,3,0]	[3,3,0]	[3,3,0]	[0,0,3]	[0,0,3]	[3,0,0]	[0,0,0]	[0,0,0]	[3,0,3]
6	1944	4455	[6,0,0]	[3,3,0]	[3,0,0]	[3,0,3]	[0,0,0]	[0,0,3]	[0,3,0]	[0,0,0]	[0,0,0]
7	2916	9072	[5,4,0]	[4,0,2]	[4,0,2]	[4,0,2]	[4,0,2]	[0,2,1]	[0,2,1]	[0,2,1]	[0,2,1]
8	1944	2298	[5,4,0]	[5,4,0]	[2,1,0]	[1,0,5]	[1,0,2]	[2,1,0]	[2,1,0]	[1,0,2]	[1,0,2]
9	5832	15390	[5,1,0]	[5,1,0]	[4,0,2]	[1,0,2]	[2,1,0]	[0,2,1]	[0,2,1]	[0,2,4]	[2,1,0]
10	1944	2622	[3,3,0]	[3,3,0]	[3,3,0]	[3,0,3]	[3,0,3]	[3,3,0]	[3,0,3]	[3,0,3]	[0,3,3]
11	324	15714	[3,0,0]	[3,0,0]	[3,0,0]	[3,0,0]	[3,0,0]	[0,3,0]	[0,0,3]	[0,0,3]	[0,3,0]

TABLE III
EXAMPLE FUNCTION TRANSFORMS

f_1 from class 8 in [8]	[0,0,1]	[0,0,1]	[0,0,1]	[0,0,1]	[1,0,0]	[0,0,1]	[1,0,0]	[1,0,0]	[0,1,0]
S_1 (spectra are in weighted order)	[2,0,4]	[0,1,2]	[0,4,5]	[2,0,1]	[2,0,4]	[4,2,0]	[0,4,2]	[0,1,2]	[2,0,1]
TRANS5: $f \leftarrow f \oplus x_2$	[2,0,4]	[0,1,2]	[2,0,4]	[2,0,1]	[0,4,5]	[0,1,2]	[2,0,1]	[4,2,0]	[0,4,2]
TRANS5: $f \leftarrow f \oplus x_2$	[0,4,5]	[4,2,0]	[2,0,4]	[0,4,2]	[2,0,4]	[0,1,2]	[2,0,1]	[0,1,2]	[2,0,1]
TRANS3A: $f \leftarrow f \oplus 1$	[5,0,4]	[0,4,2]	[4,2,0]	[2,0,4]	[4,2,0]	[2,0,1]	[1,2,0]	[2,0,1]	[1,2,0]
TRANS3B: exchange 1 and 2 for f	[5,4,0]	[2,4,0]	[4,0,2]	[0,2,4]	[4,0,2]	[1,0,2]	[2,1,0]	[1,0,2]	[2,1,0]
TRANS2A: $x_1 \leftarrow x_1 \oplus 2$	**[5,4,0]**	**[4,0,2]**	**[4,0,2]**	**[4,0,2]**	**[4,0,2]**	**[0,2,1]**	**[0,2,1]**	**[0,2,1]**	**[0,2,1]**
f_2 from class 10 in [8]	[0,0,1]	[0,0,1]	[0,0,1]	[1,0,0]	[1,0,0]	[1,0,0]	[1,0,0]	[1,0,0]	[0,1,0]
S_2	[4,0,2]	[0,2,4]	[0,2,4]	[1,0,2]	[1,0,2]	[2,4,0]	[0,2,1]	[0,2,1]	[4,0,5]
TRANS4: $x_1 \leftarrow x_1 \oplus x_2$	[4,0,2]	[0,2,1]	[0,2,4]	[0,2,1]	[1,0,2]	[0,2,4]	[4,0,5]	[2,4,0]	[1,0,2]
TRANS4: $x_1 \leftarrow x_1 \oplus x_2$	[4,0,2]	[2,4,0]	[0,2,4]	[4,0,5]	[1,0,2]	[0,2,1]	[1,0,2]	[0,2,4]	[0,2,1]
TRANS5: $f \leftarrow f \oplus x_1$	[4,0,5]	[4,0,2]	[1,0,2]	[2,4,0]	[0,2,1]	[0,2,4]	[0,2,1]	[1,0,2]	[0,2,4]
TRANS4: $x_2 \leftarrow x_2 \oplus x_1$	[4,0,5]	[4,0,2]	[0,2,1]	[2,4,0]	[1,0,2]	[1,0,2]	[0,2,4]	[0,2,4]	[0,2,1]
TRANS4: $x_2 \leftarrow x_2 \oplus x_1$	[4,0,5]	[4,0,2]	[0,2,4]	[2,4,0]	[0,2,4]	[0,2,1]	[1,0,2]	[0,2,1]	[1,0,2]
TRANS3A: $f \leftarrow f \oplus 1$	[5,4,0]	[2,4,0]	[4,0,2]	[0,2,4]	[4,0,2]	[1,0,2]	[2,1,0]	[1,0,2]	[2,1,0]
TRANS2A: $x_1 \leftarrow x_1 \oplus 2$	**[5,4,0]**	**[4,0,2]**	**[4,0,2]**	**[4,0,2]**	**[4,0,2]**	**[0,2,1]**	**[0,2,1]**	**[0,2,1]**	**[0,2,1]**

equivalence class containing the original function. The algorithm has been applied to find 11 equivalence classes for $n = 2$ and to estimate that there are some 167,275 classes for $n = 3$.

Although the number of equivalence classes will be very large for $n > 3$, the algorithm is still of utility as it can be applied to determine if two functions are in the same class and the translations to map one function to the other.

Future work will include considering the extension of the method to higher radix functions. We anticipate the challenge will be in determining how best to handle the complement operations for higher radices. Note that [8] did consider higher radix functions and that work will likely provide insights for the extension of our approach.

Our code is prototype software and we will consider possible optimizations to improve its efficiency. That is an interesting challenge since performing the necessary ternary (or higher radix) operations on a 2-valued computer is inherently complex and not amenable to the usual coding schemes commonly used to speed up code for 2-valued problems.

REFERENCES

[1] M. A. Harrison, *Introduction to Switching and Automata Theory*. New York, USA: McGraw Hill, 1963.

[2] S. L. Hurst, *The Logical Processing of Digital Signals*. London, UK: Arnold, 1978.

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

[4] J. E. Rice, *Autocorrelation Coefficients in the Representation and Classification of Switching Functions*. University of Victoria, Canada: Ph.D., 2003.

[5] N. A. Anderson, *The Classification of Boolean Functions Using the Rademacher-Walsh Transform*. University of Lethbridge, Canada: M.Sc., 2004.

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

[7] B. J. Falkowski, *Speectral Methods for Boolean and Multiple-Valued Input Logic Functions*. Portland State University, USA: Ph.D., 1991.

[8] C. Moraga, "Complex spectral logic," in *Proc. Int'l Symposium on Multiple-Valued Logic*, 1978, pp. 149–156.

[9] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Int'l Symp. on Experimental Algorithms*, 2010, pp. 178–189.

[10] A. Braeken, Y. L. Borissov, S. Nikova, and B. Preneel, "Classification of Boolean functions of 6 variables or less with respect to some cryptographic properties," in *Int'l Colloquium on Automata, Languages and Programming*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2005, vol. 3580, pp. 324–334.

[11] J. Boyar, P. Matthews, and R. Peralta, "Logic minimization techniques with applications to cryptology," *Journal of Cryptology*, vol. 26, no. 2, pp. 280–312, 2013.

[12] M. Soeken, N. Abdessaied, and G. De Micheli, "Enumeration of reversible functions and its application to circuit complexity," in *Int'l Conf. on Reversible Computation*, 2016, pp. 255–270.

[13] M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "Logic synthesis for quantum computing," *arXiv1706.02721*, 2017.

[14] J. Lv, P. Kalla, and F. Enescu, "Verification of composite Galois field multipliers over GF($(2^m)^n$) using computer algebra techniques," in *Int'l High Level Design Validation and Test Workshop*, 2011, pp. 136–143.

[15] H. E. Chrestenson, "A class of generalized Walsh functions," *Pacific J. Math*, vol. 5, pp. 17–31, 1955.

[16] M. G. Karpovsky, R. S. Stankovic, and J. T. Astola, *Spectral Logic and Its Applications for the Design of Digital Devices*. Wiley, 2008.

[17] M. S. Bespalov, "Discrete Chrestenson transform," *Problems of Information Transmission*, vol. 46, pp. 353–375, 2010.

[18] D. M. Miller and M. Soeken, "An algorithm for Walsh spectrum function classification," submitted.